The ProductNAME Project

Software Design Document

Version 2.1.5

	Prepared By	Reviewed by	Approved By
Name	SolovatSoft Client Team	Rafael Soultanov, Andrey Belyaev	ClientName
Role	Team Leader/ Team Member	Project Leader/ Team Leader/ Team Member	Client
Signature			
Date			

Revision History

Version	Revision Date	Description
2.0.1	21-Nov-2007	Added MSMQ proposal. Added old configuration files migration proposal Added COB table setup form
2.0.2	22-Nov-2007	Added functional description of Event Handling Subsystem Added class description of Event Handling Subsystem Core UML diagram and class description updated. EvensSource classes are deleted. It looks like we don't need additional plug-in classes that will implement EventSource interface to generate events described above (can be found in previous versions of the document). We can save and extract event source as an attribute of the event and process it in suitable Event Handler. Method getEventSource will return instance of an object that generated the Event. E.g. getEventSource() method called from SYSTEM Event instance will return an instance of SYSTEM class that generated the Event.
2.0.3	24-Nov-2007	SYSTEM Core class deleted. Carryable interface added. Class relations reviewed. Factory Layout class renamed to Factory Added properties and methods for classes Factory, Route, Segment, Station, SYSTEM Core UML Diagram updated and class description added.
2.0.4	28-Nov-2007	Plug-in mechanism description added Core API interfaces added
2.0.5	29-Nov-2007	Pathfinding algorithm description added Class Command renamed to Waypoint
2.0.6	30-Nov-2007	Error handling algorithm description added Object model related to pathfinding reviewed and redesigned Class description for classes related to pathfinding added
2.0.7	01-Dec-2007	Added explanation of pathfindin algorithm selection
2.0.8	06-Dec-2007	Added new version of station setup form
2.0.9	09-Dec-2007	Minor changes performed in UI section. "Close" buttons on forms has been removed
2.1.0	12-Dec-2007	UI description changed. All UI windows was divided to dialogs (modal) and forms. Main form design and layout concepts was changed.
2.1.1	21-Dec-2007	Added method "FindPlugins" to EventDispatcher class. Added Event class description. Added Methods GetXXXByID to IFactory and Iroute interfaces Added methods related to Station manipulation to route interface

Version	Revision Date	Description
		Changed method name from "StartMoving" to "Start" in ISYSTEM interface Added Incoming and Outgoing segment manipulation to the IStation interface Added data exchange algorithm description to chapter "Plug-in module mechanism".
2.1.2	27-Dec-2007	Hardware and software requirements added
2.1.3	28-Dec-2007	Radio system module diagram added
2.1.4	29-Dec-2007	Rockwell Press System Diagram added Inventory System Diagram added
2.1.5	17-Jan-2008	Deleted "Security Architecture" chapter Radio Interface Module functional specification rewritten Radio Interface Module UML diagram redesigned

Table of Contents

Revision History 2
Table of Contents 4
Overall Software Architecture 6
  Core Architecture Overview 7
    Event Handling Subsystem 7
    Factory Layout Subsystem 7
    SYSTEM Handling Subsystem 7
  External Modules Overview 7
    Press Subsystem 7
    External Data sources Subsystem 8
    Rockwell Press System Query Interface 8
    Abitrol Query Interface 8
    Inventory Query Interface 8
    Radio Interface Module 8
System Requirements 9
  Operating System Requirements 9
    Client 9
    Server 9
  Hardware Requirements 9
    Client 9
    Server 10
  Additional Software Requirements 10
    Client 10
    Server 10
  Additional System Requirements 10
Functional Specification 11
  System Core 11
    Event Handling Subsystem 12
    Factory Layout Subsystem 18
    The SYSTEM Handling Subsystem 20
    Optimal Path Finding Algorithm 22
  Radio Interface Module 26
    List of Requirements 26
    Architectural Overview 26
  Rockwell Press System Interface Module 28
  Inventory System Module 29
  Print Subsystem Module 30
UML Diagrams 32
  System Core 32
  Radio Interface 33
  Rockwell Press System Interface 34
  Print Subsystem 35
User Interface 36
  User Interface Forms 38

System Overview Form 38
Map Form 39
SYSTEM Status Dialog 40
Setup SYSTEM   Path Toolbar 41
Station Setup Form 43
Move Orders Queue Form 44
Paper Rolls Overview Form 45
Pallet Insertion Dialog 48
Staging Pallet Delivery Form 49
Staging Pallet Pickup Form 50
Empty Pallet Pickup Form 51
Pallet Status Form 52
Press Setup Dialog 53
Press Setup Form 54
Press Status Form 55
Press Status Dialog 56
Event Logging 57

Overall Software Architecture

The System is planned as a system with plug-in support. The core module provides an SYSTEM handling and plug-in modules add additional features to the software. This architecture makes the SYSTEM highly extensible and configurable.

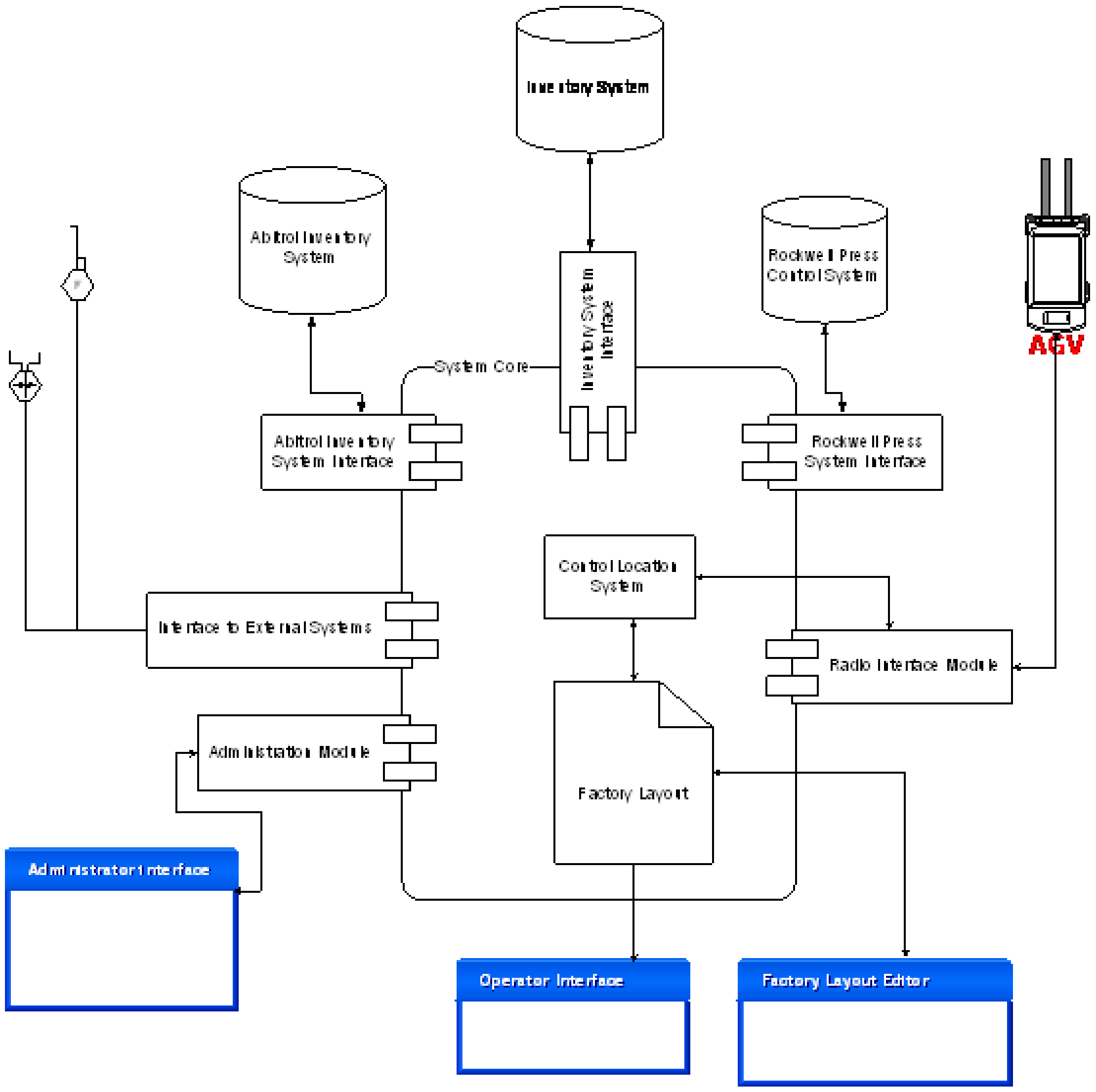


Figure 1. Overall System Architecture

The System will consist of following modules (Fig. 1):

1. System Core;
2. Inventory System;
3. Radio Interface Module;
4. Interface to External Systems;
5. Administration Module;
6. Abitrol Inventory Interface Module;
7. Rockwell Press System Interface Module;

Following applications will be developed as a stand-alone applications:

1. Factory Layout Editor
2. System Simulator

Core Architecture Overview

The Core will consist of following subsystems:

1. Event Handling
2. Factory Layout
3. SYSTEM Handling

Event Handling Subsystem

This subsystem dedicated to event handling and consist of following parts:

1. Event Queue –receives events, stores them, sorts by priority and sends to event dispatcher
2. Event Dispatcher – receives the event from Event Queue, define event type and send it to according event handler
3. Event handlers implement business logic of event handling of each event type. The handlers can call core modules via Core API. Thought business logic will be realized according to Business function diagrams.

Factory Layout Subsystem

This subsystem contains and handles all the information related to factory. Factory layout consists of Routes (wires) which are used by SYSTEM s as a controlling routes. Each route is divided to Segments. Segment is a part of route between two stations. Each segment can contain SYSTEM on itself. Each station can contain SYSTEM too.

SYSTEM Handling Subsystem

This subsystem is dedicated to assigning SYSTEM s to tasks and selecting the optimal route for SYSTEM s. It's mostly mathematical problem and it will be done by one class – Order Factory. This class (maybe with some inner additional classes) will be designed to solve the problem of selecting optimal route for SYSTEM and to generate order of commands for selected SYSTEM .

External Modules Overview

The rest of the system functionality will be provided bu external plug-in modules.

Press Subsystem

This subsystem contains and handles all the information related to press factory. Factory Layout contains Presses (and reels), rolls and pallets. All these objects interacts with SYSTEM s. Factory Layout is updated time to time to represent last information about SYSTEM location, press jobs, etc.

External Data sources Subsystem

All the data sources for system core are planned to be pluggable modules. They must implement ExternalDS interface.

Rockwell Press System Query Interface

According to documentation Rockwell Press System should be queried via sockets using special protocol.

Abitrol Query Interface

Cannot be designed because of information lack.

Inventory Query Interface

Inventory subsystem is an “internal” database designed to store information about paper rolls and pallets, its moving, using and wasting. This database is planned to be developed on MS SQL Server and it can be queried via OLE DB interface.

Radio Interface Module

For communicating with SYSTEM s radio signals are used. This subsystem will consist of the following classes:

1. RadioReceiver – sends and receives data to and from radio interface. This class is designed to work with hardware layer
2. MessageTranslator – translates data from object form to bytes for sending (and vice versa)

All the orders are from order factory should be translated by MessageTranslator and sent by RadioReceiver. All the signals from the SYSTEM are translated to events and put to EventQueue for processing.

System Requirements

The following system requirements are approximate and can be changed during system development.

Operating System Requirements

Client

Client software (E.g. operator's console) will work on following operating systems:
Microsoft® Windows® 98
Microsoft® Windows® 98 Second Edition
Microsoft® Windows® Millennium Edition
Microsoft® Windows NT® 4.0 Workstation with Service Pack 6.0a or later
Microsoft® Windows NT® 4.0 Server with Service Pack 6.0a or later
Microsoft® Windows® 2000 Professional
Microsoft® Windows® 2000 Server
Microsoft® Windows® 2000 Advanced Server
Microsoft® Windows® 2000 Datacenter Server
Microsoft® Windows® XP Home Edition
Microsoft® Windows® XP Professional
Microsoft® Windows® Server 2003 family

Server

Server software (System Core and additional plugins) will run on following operating systems:
Microsoft® Windows® 2000 Server with Service Pack 4.0
Microsoft® Windows® 2000 Advanced Server with Service Pack 4.0
Microsoft® Windows® 2000 Datacenter Server with Service Pack 4.0
Microsoft® Windows® Server 2003 family

Hardware Requirements

Hardware resources may need to be increased during production system testing. The following requirements are approximate and based on Microsoft's system requirements for running .NET applications.

Client

Required processor	Recommended processor	Required RAM	Recommended RAM	Hard Disk Space
Pentium 90 MHz*	Pentium II 450 MHz or faster	32 MB*	96 MB or higher	200 MBytes

*Or the minimum required by the operating system, whichever is higher.

Server

Required processor	Recommended processor	Required RAM	Recommended RAM	Hard Disk Space
Pentium 133 MHz*	600 MHz Pentium processor, or an AMD Opteron, AMD Athlon64 or AMD Athlon XP processor	128 MB*	256 MB or higher	3 GBytes for OS, .NET Framework and Database

*Or the minimum required by the operating system, whichever is higher.

Additional Software Requirements

Client

Name	Version
Microsoft Message Queue (Included)	2.0 and higher
Microsoft .NET Framework	1.1 (not tested on 2.0)

Server

Name	Version
Microsoft Message Queue (Included In Server versions of Windows Operation Systems)	2.0 and higher
Microsoft .NET Framework	1.1 (not tested on 2.0)
Database Server and .NET Data Provider for it	Suppose it will be MSSQL 2000 Server or later

Additional System Requirements

- Network connection to connect from client terminals to server
- Support for TCP/IP protocol
- Active Directory infrastructure is highly recommended to use the advantages of centralized security administration

Functional Specification

System Core

This module provides communication to the database, SYSTEM registers, station labels, pick and drop points as defined by the specific image of a manufacturing company in memory (factory layout), communication error handling among different modules, unsynchronized event notification for some modules in the system, and handling of error conditions in the system.
The System Core (Core) is planned as a event-driven system. The main function of the core is to dispatch messages coming from external modules (SYSTEM, Inventory, External Sensors…), by send them for handling to according module. For example, if we get message "Roll is ready" from sensor in roll preparation area, the message will be generated and sent to subsystem. This architecture gives us asynchronous process execution, small core program size (everything is done by modules) and ability to easy extend the system just by adding new message types for different modules.
The Core itself is a single class (SYSTEM Core), that handles following subsystems:

1. Event Handling
2. Factory Layout
3. SYSTEM Handling

System Core is planned to be written as a Windows Service. All communications to the core will use Event Handling subsystem as a communication channel.
Old configuration files – "Application Build File" and "Layout Builder File" (see "Specification for old systems" document) will be distributed among database and Windows registry. Database will contain following sections

1. Automatic battery change/charge maintenance (BCM) stations.
2. SYSTEM number, type mask, radio number, radio id
3. For each pick and/or drop location
    a. Location name
    b. Station number
    c. Location type
    d. Calculated minimum transfer time
    e. Load present discrete number
    f. Default pick priority
    g. Default drop location tag
    h. SYSTEM service type
    i. Queue size default limit
4. For each station
    a. stn station number
    b. Necessary discrete
    c. Signal Discrete
    d. Push station
5. Extended Drop Location Tag Definition File

6. COB tables

Registry will contain following values:
1. Company name, Installation location
2. Communications parameters
3. Move automatic priority age in minutes for each of the nine priorities
4. Host present flag
5. Discrete present flag

Event Handling Subsystem

This subsystem dedicated to event handling and consist of following parts:

Class Name	Description
EventQueue	Gets external events from Event Sources, manages them, sorts by priority and redirects to EventDispatcher
EventDispatcher	Takes messages from the queue and redirects them to an according handler. This class will be realized as a service.
EventHandler	Every event handler must implement this interface to be registered in the system. Different event handlers for different events can be plugged-in.
SYSTEM EventHandler	Class that handles the events coming from the SYSTEM
ExternalSourceEventHandler	Class that handles the events coming from external sensors, buttons, etc.
OperatorEventHandler	Class that handles the events coming from operator console
ErrorHandler	Class that handles the system errors
Event	Abstract class that encapsulate an information about event coming from the source
SYSTEM Event	Class that represent an event coming from the SYSTEM
ExternalSourceEvent	Class that represent an event coming from External Source (sensor, button, etc)
OperatorEvent	Class that represent an event coming from operator console

It is proposed that persistence layer will be developed to store event queue just in case of system crash to restore it. Thought it will be a database tables.

Functional Description
Event handling system should do the following things
1. Event Queue gets the event from event source and store it.
2. Event Queue puts the event into right place (for example, error events should be handled first).
3. Event Queue extracts the first event and sends it to EventDispatcher.
4. Event Dispatcher finds and loads an Event Handler that is able to handle given type of the event.
5. Redirect the Event object to the Event Handler.

6. Event Handler encapsulates logic that handles the event and can call methods from the other classes of the Core API.

Additional Requirements
1. Dynamically add event types
2. Dynamically add event handlers for event types
3. All these changes should be done without editing system source code

Solution
To satisfy the requirements of dynamic events and handlers addition the plug-in mechanism will be used. Windows .dll files will be used to dynamically add new Event Types and Event Handlers to the SYSTEM.

Plug-in module mechanism
The idea of plug-in modules is easy. Each module must implement interface EventHandler. This interface has a method that registers link to Factory class in the plug-in's body. The plug-in gets an access to Factory class (and instances of other classes) through public Factory methods (see System Core UML diagram, IFactory interface). For example, we can get a list of all SYSTEM s in system by calling getSYSTEM s method. This method returns a collection of instances of ISYSTEM interface. All API classes will implement corresponding interfaces, which will be contained in single .dll library. Interfaces listed below will make Core API. They will be used for additional event handlers writing. Methods of these interfaces will be the same as methods of the classes with the same name.

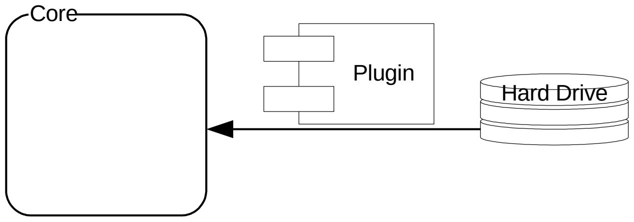
- IEventHandler
- Event
- IFactory
- IRoute
- ISegment
- ISYSTEM
- IStation
- ICarryable

Detailed description of these interfaces given in the next chapters. The common scheme of plug-in architecture sequence of actions looks like this:

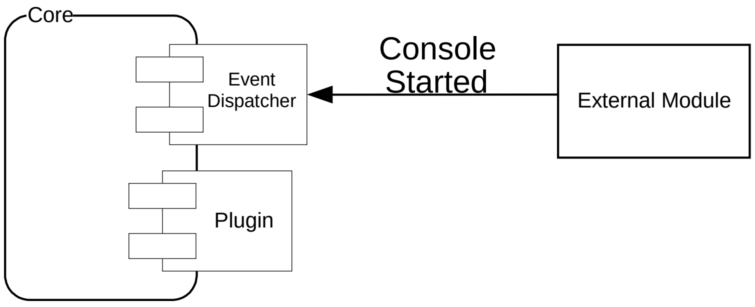
1. Event Dispatcher service starts
2. It instantiates Factory, Route, Segment, SYSTEM , Station objects and their state (loads it from database)
3. Then Event Dispatcher loads plug-in libraries and instantiates plug-in modules then registers Factory class instance in each plug-in
4. All the plugins get the access to the SYSTEM instances, Routes, Stations only through API.
5. Every plug-in can contain its own classes for work. For example Radio Interface plug-in can contain classes for working with RS-232 interface.
6. When all the plugins loaded system starts.

7. When message arrives, EventDispatcher starts corresponding EventHandler (it can be more than one handler to handle events) in separate thread. Message is handled asynchronously.
8. If external module needs to communicate with Core (needs to get some information FROM Core) it should create its own message queue. Event handler will place core answers to this queue after handling the event.
9. If there is no corresponding handler, warning message is written to Event Log
10. If external module needs factory layout for working (e.g. Factory Map Module needs the information about stations, SYSTEM s, etc.) it can be downloaded in XML format. Direct core database access should be denied to provide security and extensible architecture. The sample algorithm of Factory Map module may look like this:

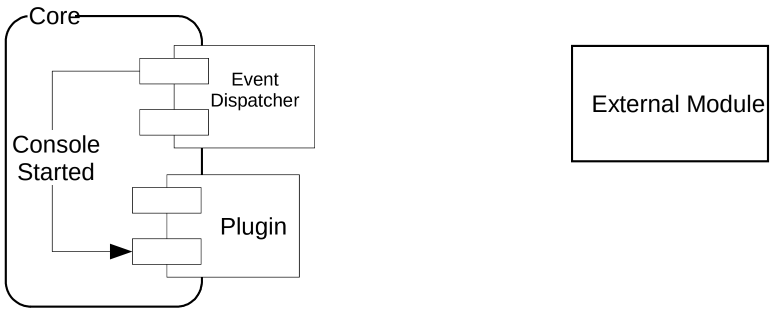
    a. System core loads event handler plugin for this module



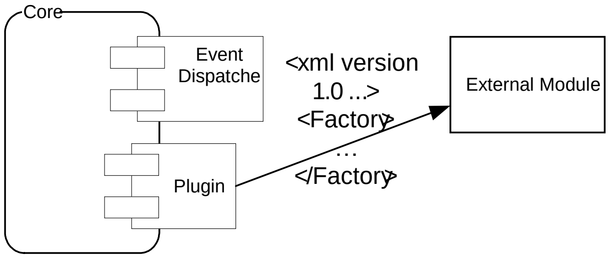
    b. User starts console on remote computer
    c. Console send the event “Console Started” to Core



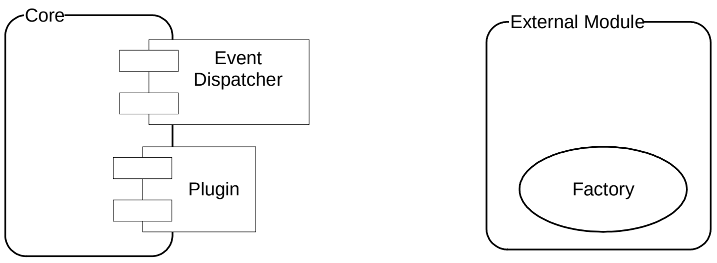
    d. Core receives the event and redirects it to event handler



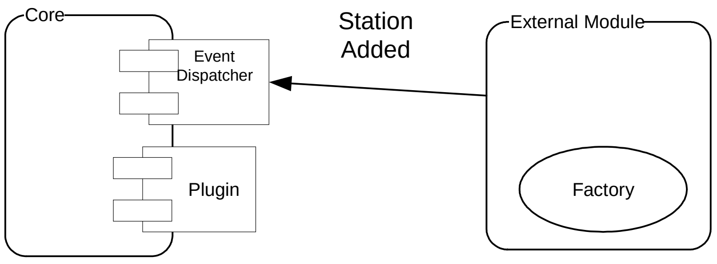
e. Event handler builds factory layout in XML form and sends it back to console using private message queue. Each client has its own event queue for answers.



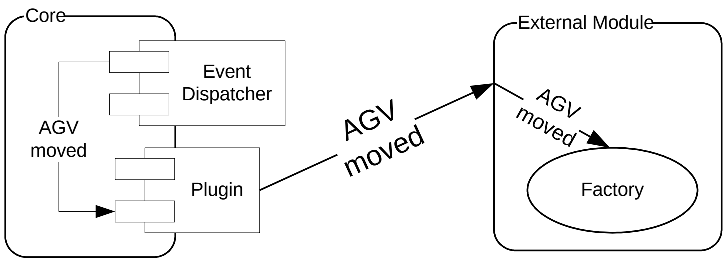
f. Console receives the layout and build its own object model (inherited from Core API interfaces) to represent it in "map" style.



g. Console can send messages about map updates (e.g. during map editing), Handler will update factory object model on server side



h. Core can send events to console about layout update from another modules (it may be non-XML protocol), so object model on client side will be updated.



Class and Interface Description

Only public methods and properties will be described. Classes that implement base interfaces (or extend base classes) will not be described unless they have some specific properties or methods.

Event Queue Class

This class can be replaced with MSMQ, which provides the same functionality.

Method Name	Description
putEvent(Event e)	This method is called by external modules to add the event to Event Queue. Parameter – instance of Event Class

Event Dispatcher Class

Will contain reference to Factory class instance. When Event handler is added to Dispatcher, it gets this reference for callbacks.

Method Name	Description
dispatchEvent(Event e)	This method is called by Event Queue when it sends the Event to Dispatcher. Parameter – instance of Event class
addEventHandler(EventHandler eh)	Register new Event Handler in Dispatcher. Event Handlers can be registered during system work. Parameter – instance of EventHandler class.
removeEventHandler(EventHandler eh)	Unregister an Event Handler. Parameter – instance of Event Handler Class to be removed.
FindPlugins(String path)	Searches plugins. Parameter – folder where plugins located

IEventHandler Interface

Method Name	Description
handleEvent(Event e)	The method is called by Event Dispatcher to handle the Event. Parameter – instance of Event class
getEventType()	Shows the type name of the Event that instance of EventHandler can handle.
getDescription()	Shows the short description of Event Handler
registerFactory()	Used for registering Factory instance in plug-in

Event Class

This class is NOT abstract. Event class must be serializable to have a possibility to send events via different transports (network, message service). Because of plug-in architecture NO Event subclassses can be send to Core. All event generators from all sources must use Event class to send events, so this class is sealed to prevent inheritance.

Property Name	Description
HIGH_PRIORITY	High-priority message type constant
NORMAL_PRIORITY	Normal-priority message type constant
LOW_PRIORITY	Low-priority message type constant

Method Name	Description
getType()	Shows the type name of the Event
getBody()	Returns event body in object form.
getSource()	Returns event source of the event.
getPriority()	Returns priority of an Event instance

MSMQ Proposal

But from the other side instead of writing event queue class Microsoft Message Queue (MSMQ) mechanism can be used. This is an analogue of “mailboxes” used in previous version of the system.
Messaging and messages provide a powerful and flexible mechanism for interprocess communication between components of a server-based application. They have a number of advantages over direct calls between components, including:

- **Robustness** — Messages are considerably less affected by component failures than direct calls between components, because messages are stored in queues and remain there until processed appropriately. Messaging is similar to transaction processing, because message processing is guaranteed.
- **Message prioritization** — More urgent or important messages can be received before less important messages, so you can guarantee adequate response time for critical applications.
- **Offline capabilities** — Messages can be sent to temporary queues when they are sent and remain that way until they are successfully delivered. Users can continue to perform operations when access to the necessary queue is unavailable for whatever reason. In the meantime, additional operations can proceed as if the message had already been processed, because the message delivery is guaranteed when the network connection is restored.
- **Transactional messaging** — You can couple several related messages into a single transaction, ensuring that the messages are delivered in order, delivered only once, and are successfully retrieved from their destination queue. If any errors occur, the entire transaction is cancelled.
- **Security** — The Message Queuing technology on which the **MessageQueue** component is based uses Windows security to secure access control, provide auditing, and encrypt and authenticate the messages your component sends and receives.

But the system must meet the following requirements to develop **MessageQueue** components:

- To see queue information in Server Explorer or to access queues programmatically, Message Queuing on your client computer must be installed.

Pros and cons for using MSMQ:

1. Development time can be decreased
2. Persistence, reliability and security problems are solved "automatically"

But

1. This infrastructure will tie our system to MS Windows Platform only.
2. Need some investigations about speed (can we guarantee particular delivery time?).
3. Need to buy Windows 2000 Server license.

Factory Layout Subsystem

This subsystem contains and handles all the information related to factory. Factory layout consists of Routes (wires) which are used by SYSTEM s as a controlling routes. Each route is divided to Segments. Segment is a part of route between two stations. Each segment can contain SYSTEM on itself. Each station can contain SYSTEM too.

Class Name	Description
IFactory	Interface that contain the information about factory layout and objects in this layout
IRoute	Route in factory layout with the dedicated frequency. Up to four routes may be in factory
ISegment	Interface that contain an information about segments (part of the route between two stations)
ISYSTEM	Represents the SYSTEM in core object model
IStation	Interface represents station on the factory route. Implementing class has three subclasses: DropLocation, PickLocation, BaseStation
ICarryable	All the object that can be carried by SYSTEM must implement ICarryable interface for easy adapting SYSTEM for specific requirements

Class and Interface description

Only public methods and properties will be described. Classes that implement base interfaces (or extend base classes) will not be described unless they have some specific properties or methods.

<u>IFactory Interface</u>

Method Name	Description
getRoutes()	Return routes present in factory layout
getSYSTEM s()	Return list of all SYSTEM s registered in factory
getSYSTEM ById()	Returns one SYSTEM instance by SYSTEM s ID.
registerSYSTEM ()	Registers an SYSTEM in factory
unregisterSYSTEM ()	Unregisters an SYSTEM in factory

Method Name	Description
addRoute()	Adds additional route in factory layout
removeRoute()	Removes route from factory layout

IRoute Interface

Method Name	Description
getID()	Returns route number
getFrequency()	Returns frequency value of the route
getSegments()	Returns list of segments route consists of
addSegment()	Adds a segment to route
removeSegment()	Removes a segment from route
getStations()	Returns stations array, which are located on the route
getStationById()	Returns one stations by its label
addStation()	Adds station to the route
removeStation()	Removes station from the route

ISegment Interface

Method Name	Description
getLength()	Returns the segment length
getStartStations()	Returns a start station of the segment
getEndStation()	Returns an end station of the segment
setStartStation()	Sets the start station of the route
setEndStation()	Sets the end station of the route
setRoute()	Sets the route for a particular segment (may be needed for route development tasks)
getRoute()	Returns the reference to Route that a segment belongs to
getSYSTEM s()	Returns a list of SYSTEM s that are on this route

ISYSTEM Interface

Method Name	Description
getLoad()	Returns an instance of Carryable – current load of an SYSTEM
setLoad()	“Loads” an SYSTEM with the instance of Carryable
getCurrentStation()	Returns station that an SYSTEM moves to.
getId()	Returns an SYSTEM number
getSegment()	Returns the segment that an SYSTEM is moving along
getDropLocation()	Returns a drop location assigned to an SYSTEM
setDropLocation()	Assigns drop location for an SYSTEM
getPickLocation()	Returns a pick location assigned to an SYSTEM
setPickLocation()	Assigns pick location to an SYSTEM
getParkLocation()	Returns a park location of an SYSTEM
setParkLocation()	Assigns a park location to an SYSTEM
getStatus()	Returns SYSTEM current status

Method Name	Description
setStatus()	Sets a particular status to an SYSTEM
getAction()	Returns current action that an SYSTEM executes
getMoveNumber()	Returns move number of an moving order
start()	Starts SYSTEM

<u>SYSTEM Class</u>

Method Name	Description
setOrder()	Protected. Sets the move order to an SYSTEM
getOrder()	Protected. Returns an SYSTEM ’s move order

<u>IStation Interface</u>

Method Name	Description
getIncomingSegments()	Returns a list of incoming segments for this station
getOutgoingSegments()	Returns a list of outgoming segments for this station
addIncomingSegment()	Adds incoming segment for the station
addOutgoingSegment()	Adds outgoing segment for the station
getSYSTEM ()	Returns an SYSTEM which resides on a station
getId	Returns a station Id (label)
getStatus()	Returns station status
setStatus()	Assigns a status to a station
getLoad()	Gets a Carryable instance – something that lays on a location
setLoad()	Puts a Carryable instance to a location

<u>ICarryable Interface</u>

Method Name	Description
setSYSTEM ()	“Loads” Carryable to an SYSTEM
getSYSTEM ()	Returns the reference to an SYSTEM that carries this object
setStation()	“Puts” the object to a station
getStation()	Returns the reference to a station that object lays on

The SYSTEM Handling Subsystem

This subsystem is dedicated to assigning SYSTEM s to tasks and selecting the optimal route for SYSTEM s. It’s mostly mathematical problem and it will be done by one class – Order Factory. This class (maybe with some inner additional classes) will be designed to solve the problem of selecting optimal route for the SYSTEM and to generate order of commands for selected SYSTEM .

Class Name	Description
OrderFactory	Class that generates orders and assigns them for SYSTEM s. It

Class Name	Description
	finds SYSTEM to execute, finds the shortest route, assigns order for SYSTEM and puts it into OrderQueue – array of orders that are waiting of execution, sort them by priority and call SYSTEM class methods to assign order. OrderFactory is also contains an array of pre-computed paths.
Path	Set of segments from start station to goal station.
Order	Set of waypoints (path) for SYSTEM and actions to execute on each waypoint.
Action	Single action for the SYSTEM to execute. Contains station where action should be executed and action description in form “CAL table id + modifier”. Each action has its state (waiting, executing or completed), time to come and action duration. Used inside Order class only.

Class and Interface description

Only public methods and properties will be described. Classes that implement base interfaces (or extend base classes) will not be described unless they have some specific properties or methods.

<u>OrderFactory Class</u>

Method Name	Description
setStart()	Sets start point of route
setGoal()	Sets goal point of route
generateSYSTEM Order()	Generates the order for the SYSTEM if start and goal points are set. After generation nullifies start and goal points
getOrderQueue()	This method allows to get an access to orders process list

<u>Path Class</u>

Method Name	Description
getSegments()	Returns the segments that a path consists of
addSegment()	Adds one segment to a path.
removeSegment()	Removes a segment from a path.
getLength()	Returns the length of a segment. This method does not count collision resolve time.
getStations()	Returns a list of a stations that belongs to a path

<u>Order Class</u>

Method Name	Description
getSYSTEM ()	Gets the SYSTEM this order assigned to.

Method Name	Description
setSYSTEM ()	Protected. Assigns the order to SYSTEM. Cannot be accessed from outside.
getPath()	Returns a path assigned to the order.
getActions()	Returns a list of actions that are assigned for this order. Each action can be changed then.
addAction()	Adds an action for a station. Only actions for stations in the path can be added.

Action Class

Method Name	Description
getCalId()	Returns CAL table number that an action associated to
setModifier()	Sets CAL modifier
getModifier()	Returns CAL modifier for an action
getState()	Returns action state (wait, executing, completed)
getStation()	Returns station the action should be executed on
setStation()	Protected. Assigns the station for the action
getDuration()	Returns duration of an action
getComingTime()	Returns a time when the SYSTEM will come to a station to execute an action. Needed for collision prevention.
setComing Time	Protected. Changes the SYSTEM's coming time to a station.

Optimal Path Finding Algorithm

Let's make some assumptions

1. Each robot has an assigned goal, and each robot knows its start and goal positions;
2. Robots have a pre-defined path system;
3. Robot can walk out of path, but in pre-defined places only (on stations)
4. Robots cannot communicate to each other.
5. In case of collision robot stops and waits for commands
6. All robots has a constant speed
7. Robot can stop and do some work only on stations
8. All path are unidirectional
9. Robot does not have reverse speed
10. In the end of work robot goes to particular place for charging

The aim of optimization is to minimize the time for loaded SYSTEM moving. The aim of free robot path optimization is to minimize collisions with loaded SYSTEM s. It can be not optimal in terms of time, but it will minimize the time for loaded robots.

There are a number of algorithms for finding paths in dynamical environment: A*, D* and its modifications, adapted genetic algorithm, ant-based pathfinding, neural networks can be adapted.

Neural networks and ant-based pathfinding are not optimal for this problem, because they are too interial. They need too much time for teaching (of course they can be teached using simulators) and in case of Factory Layout changing they will make a big number of errors. Moreover, these algorithms are quite hard to implement.
Genetic algorithm as also hard to implement on program language and they are less “natural” from the poit of clearness.
A* and D* are heuristic algorithms and are not computationally expensive, but they are good for partially unknown environments or big graphs with a big number of nodes and arcs.
Exact algorithms such as Dijkstra or Floyd algorithms computationally more expensive, but they generate exact results for well-defined situations. In case of good implementation exact algorithms can be quite fast (AMD K6-2 400MHz RAM 64M, graph nodes = 100000, arcs = 200000 finding the shortest path takes 0.4 seconds (AGraph library)), and they are easier to implement than heuristic or neural algorithms. So, it is proposed than exact algorithms will be used.

All SYSTEM s has a state, so has priorities (less number – higher priority)

1. Moving to drop
2. Moving to pick
3. Moving to charge
4. Moving to park

States
1. Parked
2. Picking
3. Dropping
4. Charging

are “static” states and do not have an impact to the pathfinding algorithm.

If SYSTEMs has the same status, the SYSTEM that have got the task earlier, has a higher priority. If the task was given simultaneously, the SYSTEM with the shortest rest of the way has a higher priority. Prioritizing needs for deadlock prevention.

All the calculations will be made on digraph with weighted arcs (route segments) and nodes (stations). The weight of arc is the time needed for the SYSTEM to go along the arc. The weight of node is the time that SYSTEM should stay on this node. A weight of a node is a dynamic value (depends on time), and a weight of an arc – static value.

As it can be understood from documentation, all of the routes are the routes from pick locations to drop locations, from drop locations to service locations and from service locations to pick locations. So the number of possible paths is a limited value. The first idea of increasing speed of calculations is precomputing all “typical” paths and store them in memory (possibly in database in case of very large number of paths). So, we have a limited number of paths stored in memory with precomputed length (length mean time to go from start location to goal location in case of other robots absence). All the paths in this path list are sorted by length.

Initially there are four sets of stations: *P* – pick locations, *D* – drop locations, *S* – service (charge) locations, *B* – base locations. We add fifth set of location – *I* – intersections of segments. Intersections are important, because they are used more intensively than base stations and intersection is always a potential bottleneck in any traffic systems.

The aim of the algorithm is minimizing value $T(R_j) = \sum_{i=1}^{N} C_i + \sum_{k=1}^{M} N_k$, where $T(R_j)$ - time for SYSTEM number j. $\sum_{i=1}^{N} C_i$ - the cost of the path without collisions ("static" cost), $\sum_{k=1}^{M} N_k$ - the time that SYSTEM must stay on Station number k. Let's make the algorithm (without loss of generalization) for delivering roll from pick location S1 to drop location G1. This algorithm uses the "velocity-path" decomposition, "path" part ($\sum_{i=1}^{N} C_i$) is precomputed and "velocity" part ($\sum_{k=1}^{M} N_k$) should be calculated.

1. Set up start point and goal point (S1 $\in$ *P* and goal point G1 $\in$ *D*)
2. Select the shortest path from the precomputed paths
3. loop
4. Find the possible collisions. Collision occurs if two SYSTEM times are equal on the same station.
5. If potential collision is identified the station before station on the route where collision is identified gets a weight which equals the time the SYSTEM should stay to resolve collision according to SYSTEM priorities.
6. If potential collision node ($N_{coll} \in I$) try to set weight for previous node (N) on the route that belongs to *B* set.
7. When all nodes are weighted, the cost of the path recalculated.
8. If selected path cost less than "static" path cost of the next precomputed path in a path list (or it's a last path) take this path as a shortest path and go to step 10
9. Else select next path from path list and go to step 3
10. Assign the path found to SYSTEM

Data structures

For storing graph used in algorithm adjacency list will be used. Every node (station) has a link to its neighbors, so we have description of the full factory layout. From the other side, each route segment has a link to its start and end stations, so, the factory layout graph is stored as an array of nodes (stations), array of arcs (segments) and as a adjacency list (each station has an array of segments it belongs, so it has a list of near stations). Adjacency matrix may be used in case of performance problem with the list. Now there's no reason to use this representation because it looks like there are more than 50% of the nodes have single incoming arc and single outgoing arc, so the matrix will be strongly rarefied. Moreover, object representation gives more "natural" way for manipulating graphs, so it will be easier to implement algorithms.

Error conditions handling

In case of SYSTEM breakdown the system will find all the SYSTEM s that using the segment where breakdown occurs and stops them on the nearest station. Then the system will try to re-plan paths of the SYSTEM s from the stop point to goal point. The order of the re-planning – from the nearest to breakdown to the farthest. If there's no alternative path the SYSTEM will be stopped until broken SYSTEM will be taken off from the segment. During the breakdown segment's length is changing to infinity number to exclude it from path planning. When broken SYSTEM will be moved off the path, the segment's length will be changed to its previous value. If some SYSTEM 's were moving to longer paths it will be attempted to re-plan their paths to redirect them to more optimal path. Dynamic path re-planning will be implemented using Dijkstra algorithm.
The sequence of steps in case of error condition will look like this:

1. Define segment when error occurred
2. Mark all segments from breakdown backward to nearest crossing as locked
3. Create list of SYSTEM s that located on these segments
4. Stop these SYSTEMs.
5. For each SYSTEM in list starting from the nearest to breakdown
6. Modify the SYSTEM's action list to move it as close as possible to broken SYSTEM and stop. Mark the segment preceding to breakdown segment as locked.
7. Start the SYSTEM
8. If SYSTEM list is not empty move to step 5
9. For each SYSTEM located before locked part starting from the nearest to breakdown
10. Find alternative path to goal point using Dijkstra algorithm and static weights of the segments
11. If the path found, recalculate velocity profile for the SYSTEM and go to step 10
12. Else modify the SYSTEM's action list to move it as close as possible to broken SYSTEM and stop. Mark the segment preceding to breakdown segment as locked.
13. Start the SYSTEM
14. If SYSTEM list is not empty move to step 9

The sequence of steps when broken SYSTEM taken off from the segment

1. Create list of SYSTEM s that may use this segment.
2. For each SYSTEM closest to the segment.
3. Find path that is more short than current.
4. If path found calculate velocity profile.
5. If the path still has smaller weight reassign SYSTEM to this path.
6. Else let the SYSTEM stay on previous path.

Requirement to recalculation algorithm: total recalculation time for one SYSTEM must be less than 0,5 seconds.

Task assignment algorithm

When the system gets a task it should execute the following sequence of actions:

1. Create a list of SYSTEM s which are not carrying any load and not going for charging.

2. For all free SYSTEM s find the closest SYSTEM for executing the task. First try to find if the situation is “standard” (precomputed), else generate the route using Dijkstra algorithm.
3. Then velocity profile for the SYSTEM that has the shortest static route is generated.
4. The shortest route is selected from all routes found on step 2.
5. If generated route (with velocity profile) is still the shortest then the SYSTEM is selected for task execution
6. Else select the shortest route without velocity profile and go to step 3
7. SYSTEM assigned to the path that have been generated then starts moving
8. If during the SYSTEM’s moving another SYSTEM becomes free (and it does not needed to be charged), the system calculates if it is closer to task point than currently assigned SYSTEM.
9. If yes, the path for new SYSTEM generated (with velocity profile generation).
10. If this path is shorter than currently SYSTEM has, the task is assigned to new SYSTEM
11. Previous SYSTEM becomes free and goes to parking point (or can be assigned to another task)

So, it is proposed to use exact Dijkstra algorithm for computing the shortest path and a number expert rules for collision and error conditions resolving.

Radio Interface Module

Module responsible for communication with radio subsystem. This module is responsible for support of constant stream of information to be sent and received using the radio interfaces. Will notify the core system module when completing tasks for sending messages. It will be programmed to scan the condition of the main modules of the system automatically (if has no first priority tasks for execution). This module is responsible for reducing waiting time and all radio interface activity.

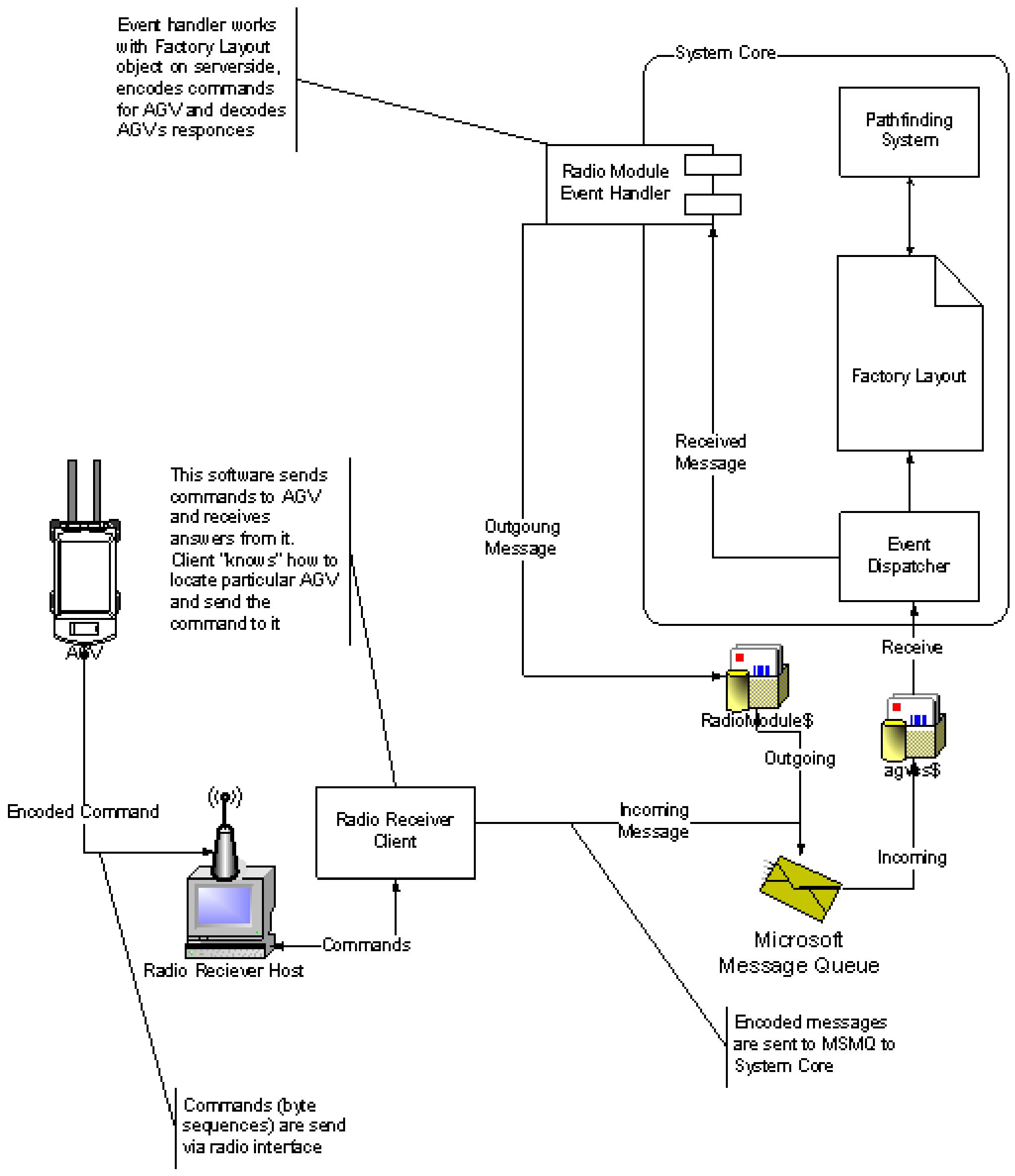
List of Requirements

1. Communication errors handling
2. Isolation from low-level interface (RS-232 or others)
3. Make status poll automatically

Architectural Overview

The module can be divided to two parts: a Radio Receiver Client and a Radio Module Event handler. The Radio Receiver Client instances located on communication hosts, there can be up to four Radio Receiver Client instances (according to old system specification). The Radio Module Event Handler is designed as core plug-in and resides on host where System Core located.
The Radio Receiver Client (Just “Client” from here) just sends and receives arrays of bytes. It “knows” how to send command to particular SYSTEM in terms of communicating protocol used. Each instance of the Client has a list of SYSTEM s it controls. Each SYSTEM in list has a command queue. If there are no commands come from System Core, the Client polls SYSTEM s automatically by generating a command and putting it into command queue for particular SYSTEM and sends received status reports to the Core. All instances of the Client use one (common) message queue to send events to the System Core. Each Client instance use its own

message queue to get byte array to send from the Core. All byte arrays are sent via RS-232 interface. When a message arrives, the Client creates message and puts it to MSMQ. The message contains encoded byte array.
The Radio Module Event Handler, from the other side, receives messages from all Clients, decodes them and updates factory layout according to the status received. It also contains a list of radio Clients and information about which SYSTEM is connected to each Radio Module Client. If another module starts SYSTEM (e.g. SYSTEM can be started from Operator Interface) or changes its status, it notifies the Core by creating event and putting it to MSMQ. Radio Module Event Handler gets the notification and generates command for SYSTEM. Then this command is "wrapped" into message and put to the Client message Queue.


Event handler works with Factory Layout object on serverside, encodes commands for AGV and decodes AGV's responces
System Core
Pathfinding System
Radio Module Event Handler
Factory Layout
Received Message
Outgoung Message
Event Dispatcher
This software sends commands to AGV and receives answers from it. Client "knows" how to locate particular AGV and send the command to it
AGV
Receive
RadioModule$
agvs$
Outgoing
Incoming Message
Incoming
Encoded Command
Radio Receiver Client
Microsoft Message Queue
Commands
Radio Reciever Host
Encoded messages are sent to MSMQ to System Core
Commands (byte sequences) are send via radio interface

Class Name	Description
RadioReceiver	Class responsible for sending and receiving information via radio channel. It works with hardware. There can be up to four

Class Name	Description
	RadioReceiver instances to service up to four radios.
MessageTranslator	Class translates messages from object representation to byte arrays.
RadioHandler	Plug-in class that handles events coming from RadioReceiver and sends commands to SYSTEM via the same RadioReceiver.

Class and Interface description

Only public methods and properties will be described. Classes that implement base interfaces (or extend base classes) will not be described unless they have some specific properties or methods.

RadioReceiver Class

Method Name	Description
SendBytes()	Sends byte array to particular SYSTEM
ReceiveBytes()	Receives byte array from radio interface

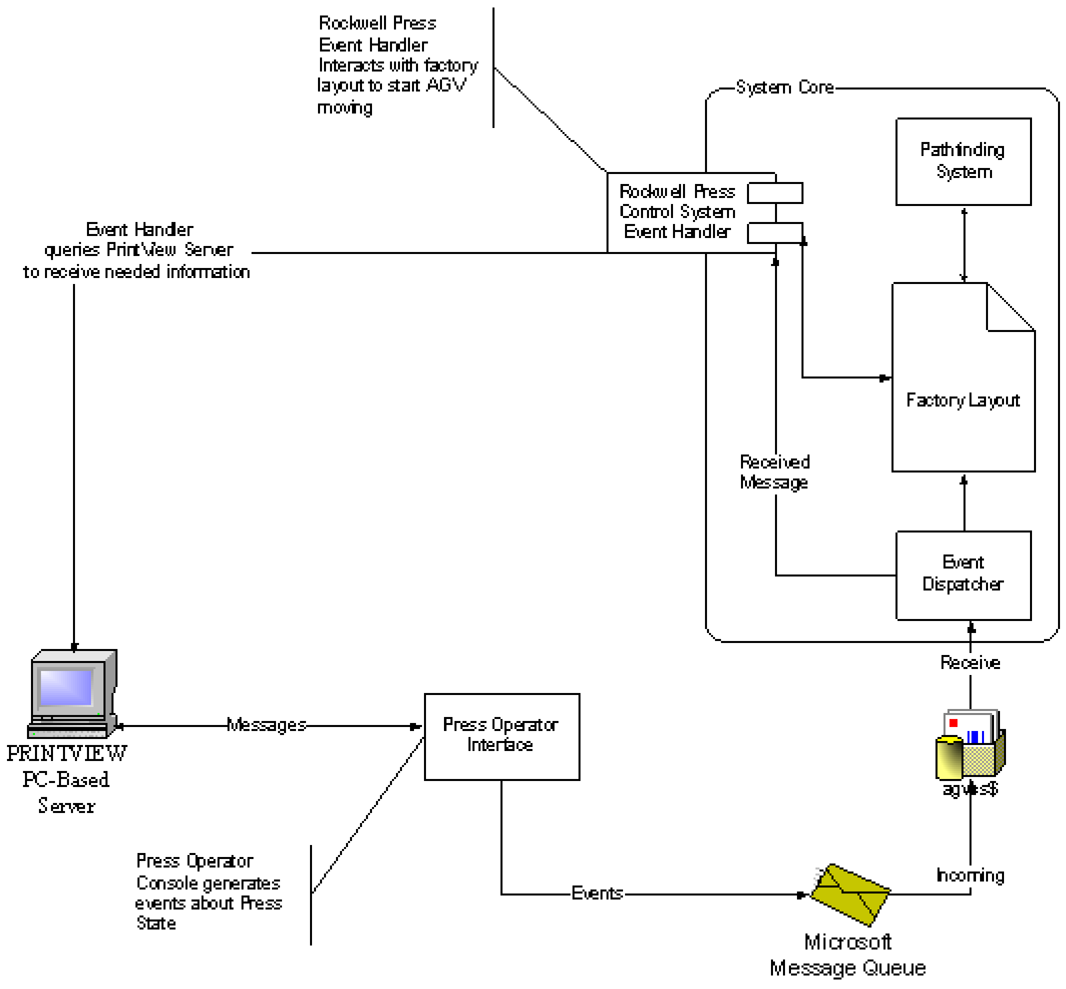
MessageTranslator Class

Method Name	Description
TranslateOrder()	Translates order for SYSTEM to byte array
TranslateEvent()	Translates incoming byte arrays to system events

MessageTranslator Class
This class implements IeventHandler interface, so their descriptions are the same.

Rockwell Press System Interface Module

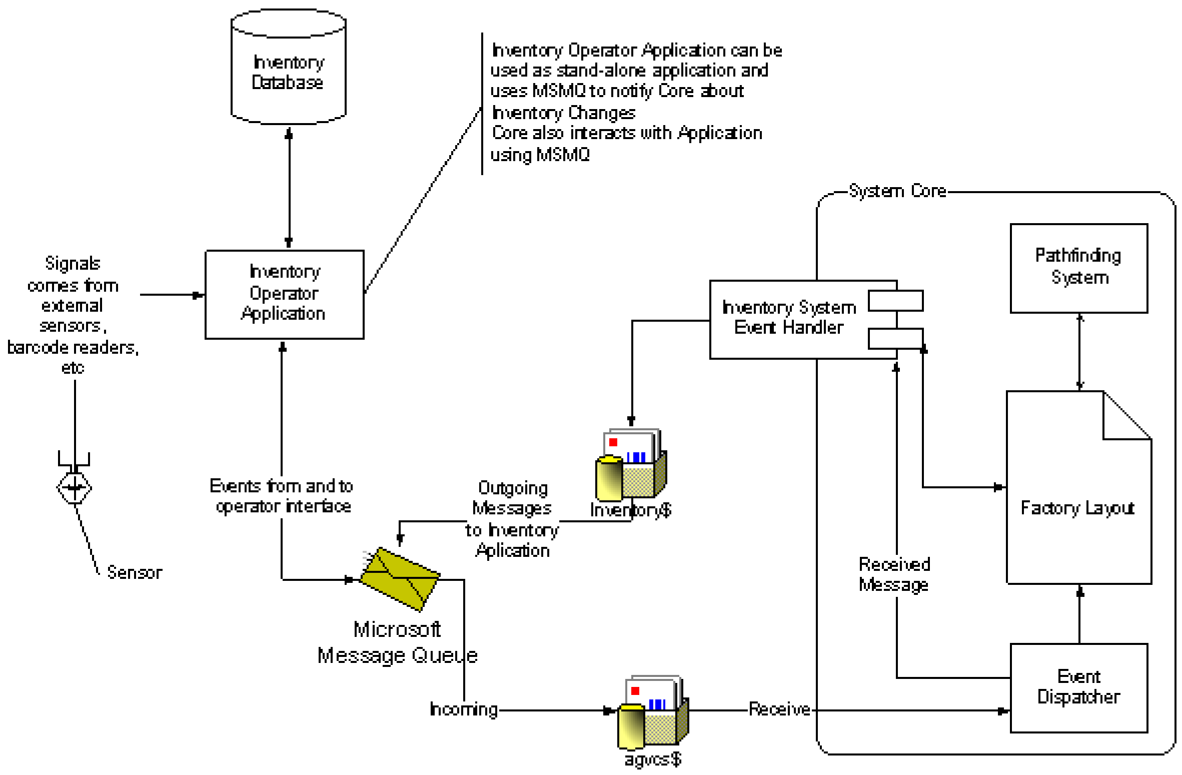
According to documentation Rockwell Press System should be queried via sockets. When it is needed to start press job, QueryData() method of class RockwellDS is called from the core, then queryPress() method is called. This method generates query message, send it to press and receives an answer. Then results are returned as a result of QueryData() method execution.


Rockwell Press Event Handler Interacts with factory layout to start AGV moving
System Core
Pathfinding System
Rockwell Press Control System Event Handler
Event Handler queries PrintView Server to receive needed information
Factory Layout
Received Message
Event Dispatcher
Receive
PRINTVIEW PC-Based Server
Messages
Press Operator Interface
agvs$
Incoming
Press Operator Console generates events about Press State
Events
Microsoft Message Queue

Class Name	Description
RockwellPress	Class that represents press to be queried. Each press has links to two instances of RockwellServer class – primary and backup.
RockwellServer	This class contains information about servers: IP address, name, etc
RockwellMessage	This class represents message to be send (and received) to (and from) rockwell server.

Inventory System Module

Inventory system object model contains of classes represented in factory layout model. All these classes will be mapped to Inventory Data Model described in corresponding document.


Inventory Database
Inventory Operator Application can be used as stand-alone application and uses MSMQ to notify Core about Inventory Changes
Core also interacts with Application using MSMQ
System Core
Signals comes from external sensors, barcode readers, etc
Inventory Operator Application
Pathfinding System
Inventory System Event Handler
Factory Layout
Sensor
Events from and to operator interface
Outgoing Messages to Inventory Aplication
Inventory$
Received Message
Microsoft Message Queue
Incoming
agvcs$
Receive
Event Dispatcher

Class Name	Entity Name
Press	Presses
Reel	Reels
Roll	Rolls, Roll_Categories, Roll_Qualities, Manufacturers, Paper_Colors
Link between Reel and Roll	Roll_On_Reels
Pallet	Pallets, Pallet_status, Statuses, Advertisers, Rack_Locations
Insert	Inserts

Print Subsystem Module

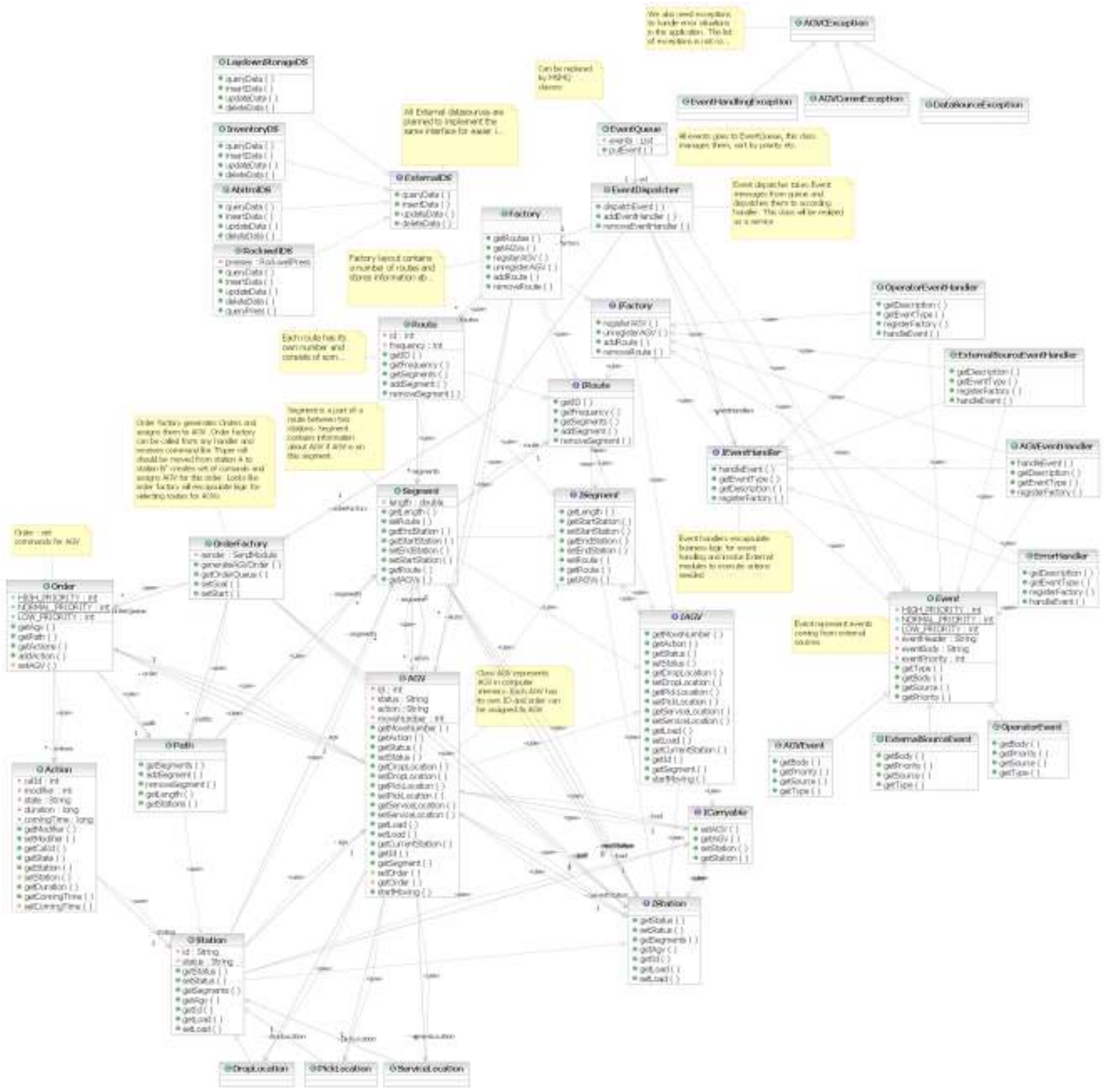
Inventory system object model contains of classes represented in factory layout model. All these classes will be mapped to Inventory Data Model described in corresponding document.

Class Name	Entity Name
PressJob	Contains the information about rolls needed for press job
PressJobInfo	Contains the Reel-Roll information about how many rolls needed on which press reel.
Roll	Contains the information about press rolls
Pallet	Class represents pallets in the system. Pallets are used in press job.
Insert	Each pallet contains a number of inserts. This

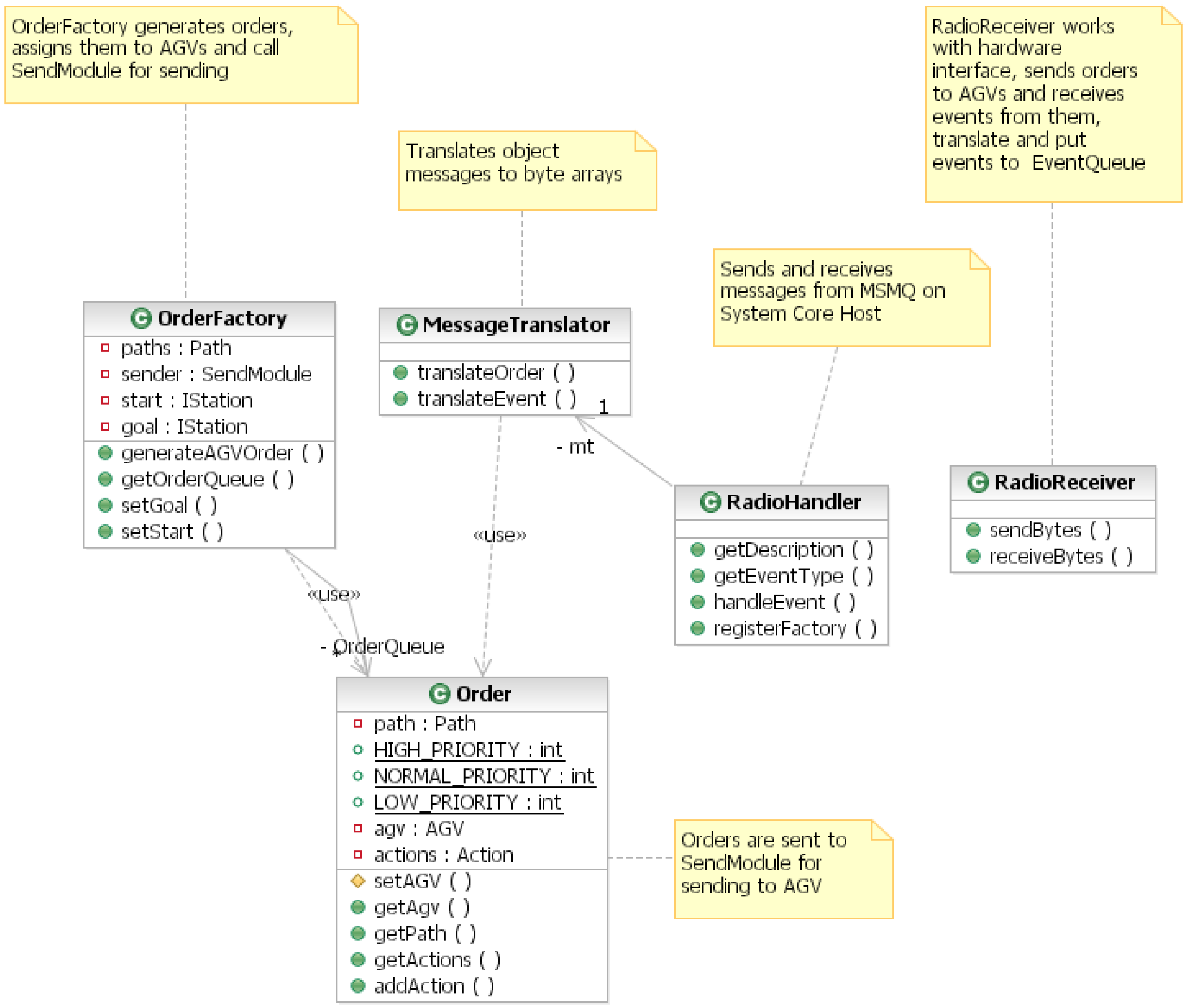
Class Name	Entity Name
	class represent one insert instance
Press	Class represent press in factory
Reel	This class represents reels on press.
PressJob	Contains the information about rolls needed for press job
PressJobInfo	Contains the Reel-Roll information about how many rolls needed on which press reel.
Roll	Contains the information about press rolls
Pallet	Class represents pallets in the system. Pallets are used in press job.

UML Diagrams

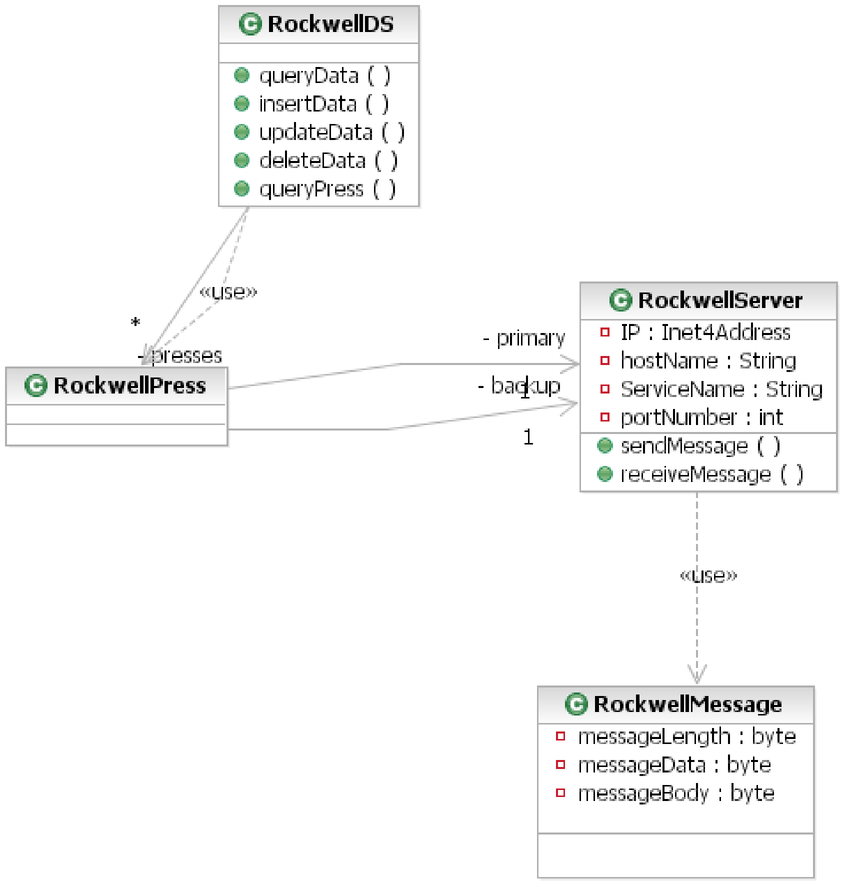
System Core



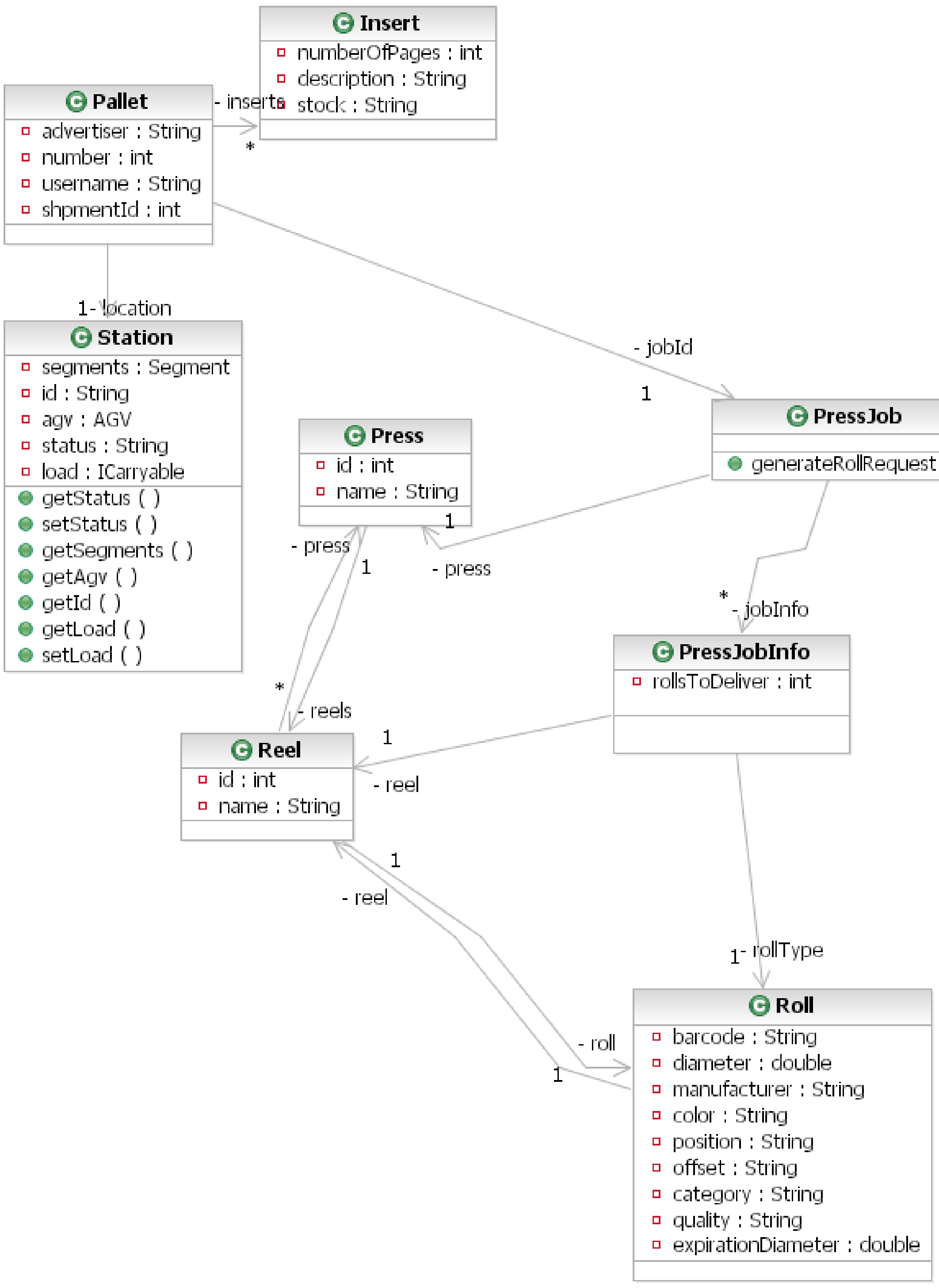
Radio Interface



Rockwell Press System Interface



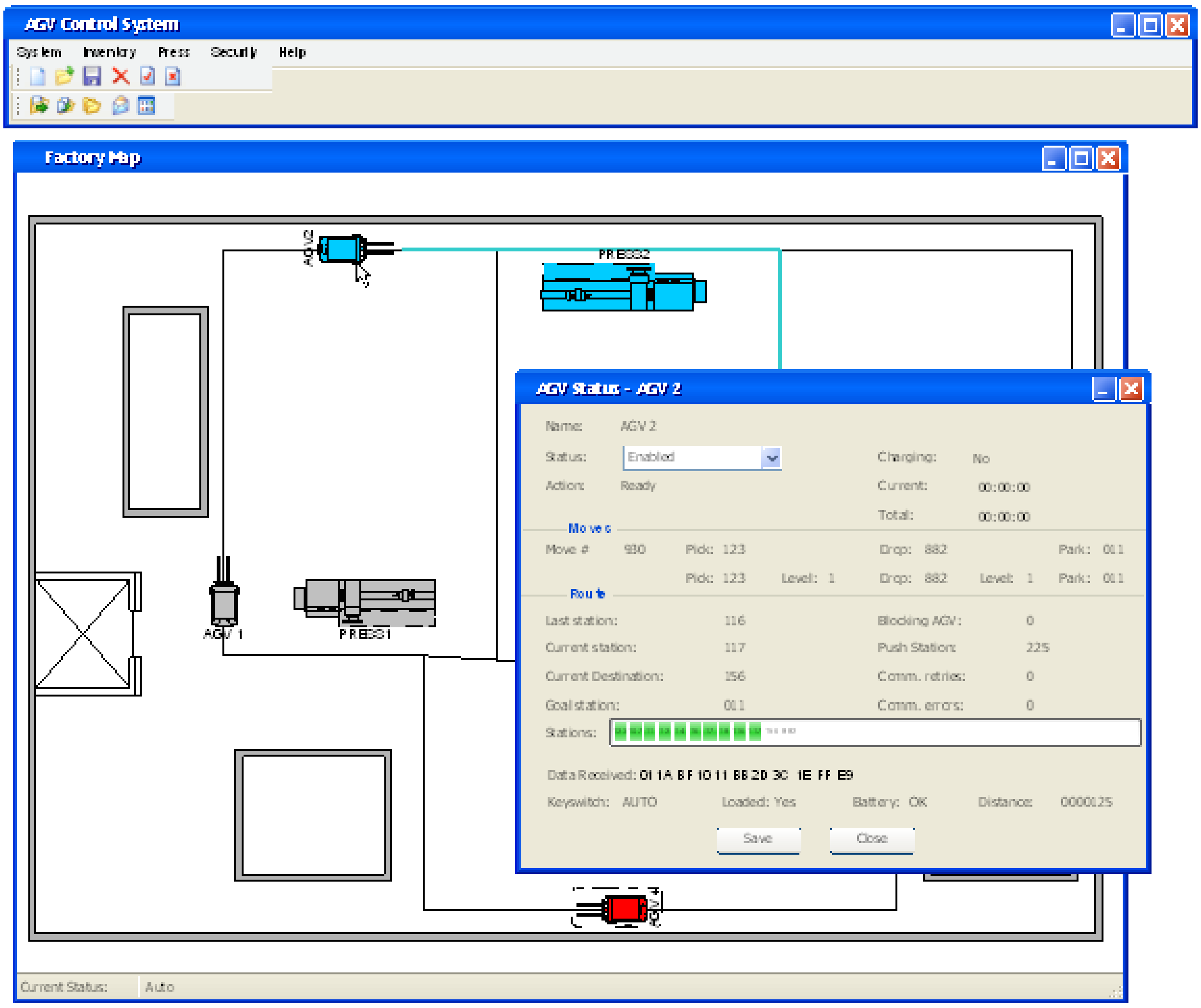
Print Subsystem



User Interface

The interface of the new SYSTEM C system will utilize all the features of rich windows interface: drop-down lists with spellchecking, drag-and-drop mechanism, customizable toolbars, hot keys, etc. Supposed that there will be map representation of system status, and user will get status of any object on the map (SYSTEM , Station, Press) just by clicking on it. Status of each object will also be highlighted by color for easier information perception. But from the other hand there will be a set of keyboard-optimized forms for quick and mass input of data.
Some forms will reproduce old console forms (user will not be needed to restudy), but even these forms will utilize all Windows UI features.

This application will use "Project" (according to Microsoft's "Official Guidelines for User Interface Developers and Designers") window management model. It will be main window with menu bar and toolbars and a number of independent windows that can be managed independently. Each window will have its own button on windows taskbar



AGV Status - AGV 2

Name: AGV 2
Status: Enabled
Action: Ready
Charging: No
Current: 00:00:00
Total: 00:00:00

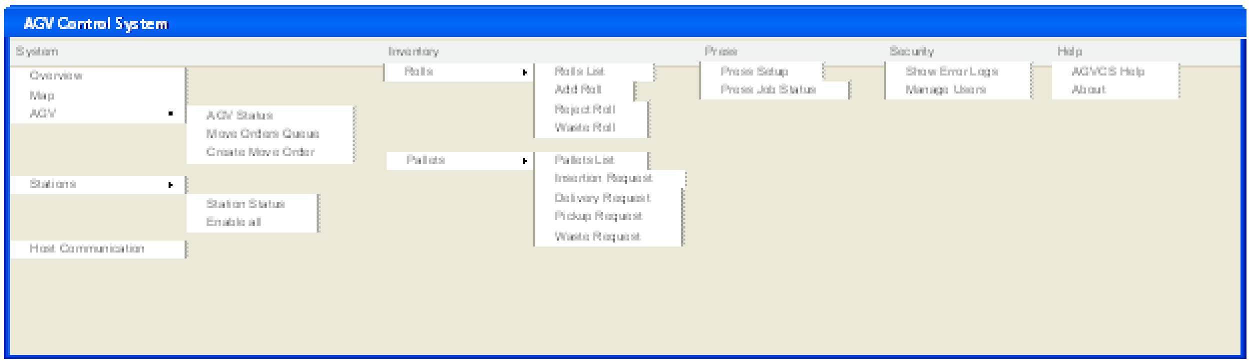
Moves
Move # 930 Pick: 123 Drop: 882 Park: 011
Pick: 123 Level: 1 Drop: 882 Level: 1 Park: 011

Route
Last station: 116 Blocking AGV: 0
Current station: 117 Push Station: 225
Current Destination: 156 Comm. retries: 0
Goal station: 011 Comm. errors: 0
Stations:

Data Received: 01 1A BF 10 11 BB 20 3C 1E FF E9
Keyswitch: AUTO Loaded: Yes Battery: OK Distance: 0000125

Current Status: Auto

Menu structure was developed according to business process description and forms in menu will be developed to help users to do their routine operations more efficiently.

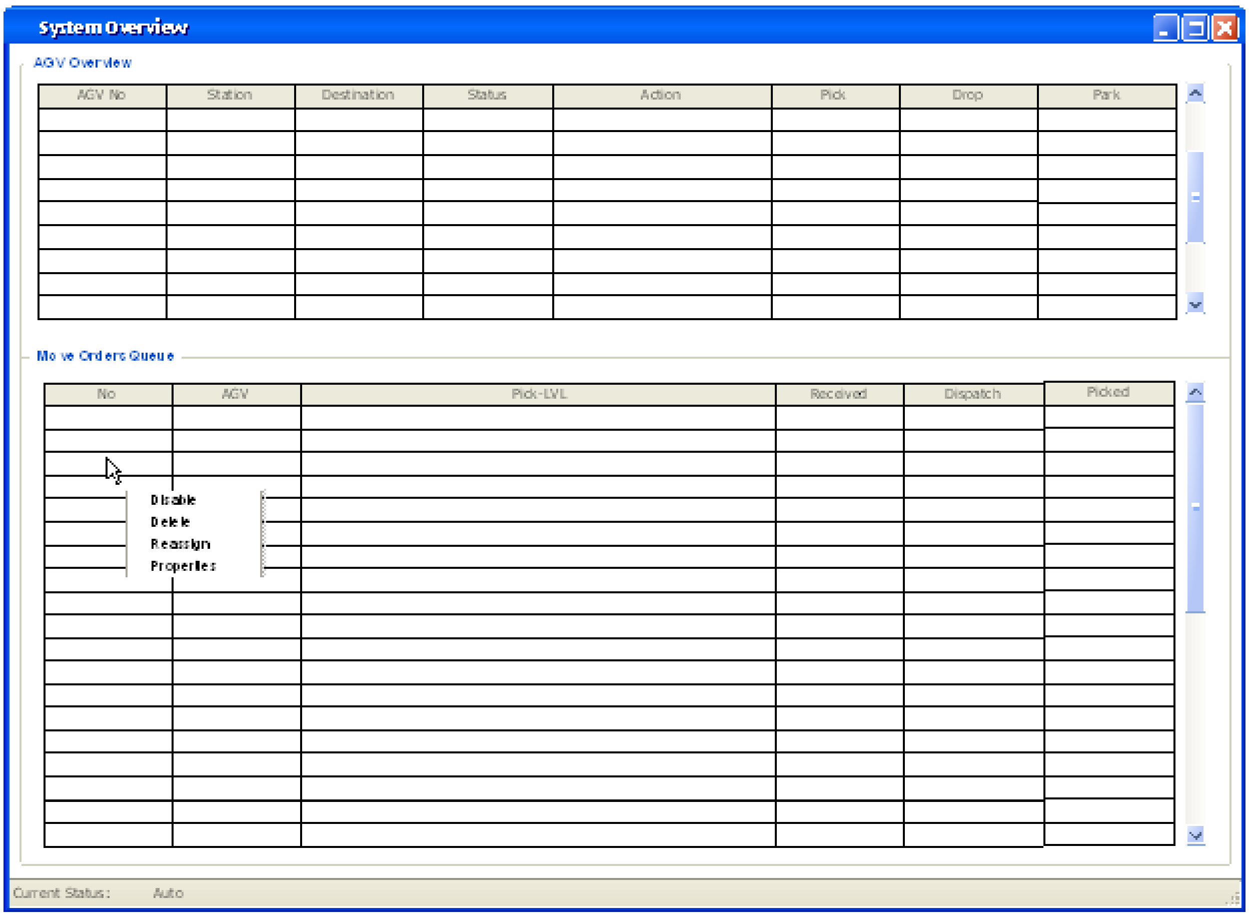


User interface will interact with the SYSTEM via Events to isolate interface implementation from system. The main reason to do this is isolation of UI implementation from system implementation. It gives us an ability to use hardware terminals or UI implemented under different OS.

User Interface Forms

System Overview Form

This form represents system status in old, table-view form. By clicking on SYSTEM or Move Order context-depended actions (e.g. SYSTEM disabling or Move Order removal) can be executed.



AGV No	Station	Destination	Status	Action	Pick	Drop	Park

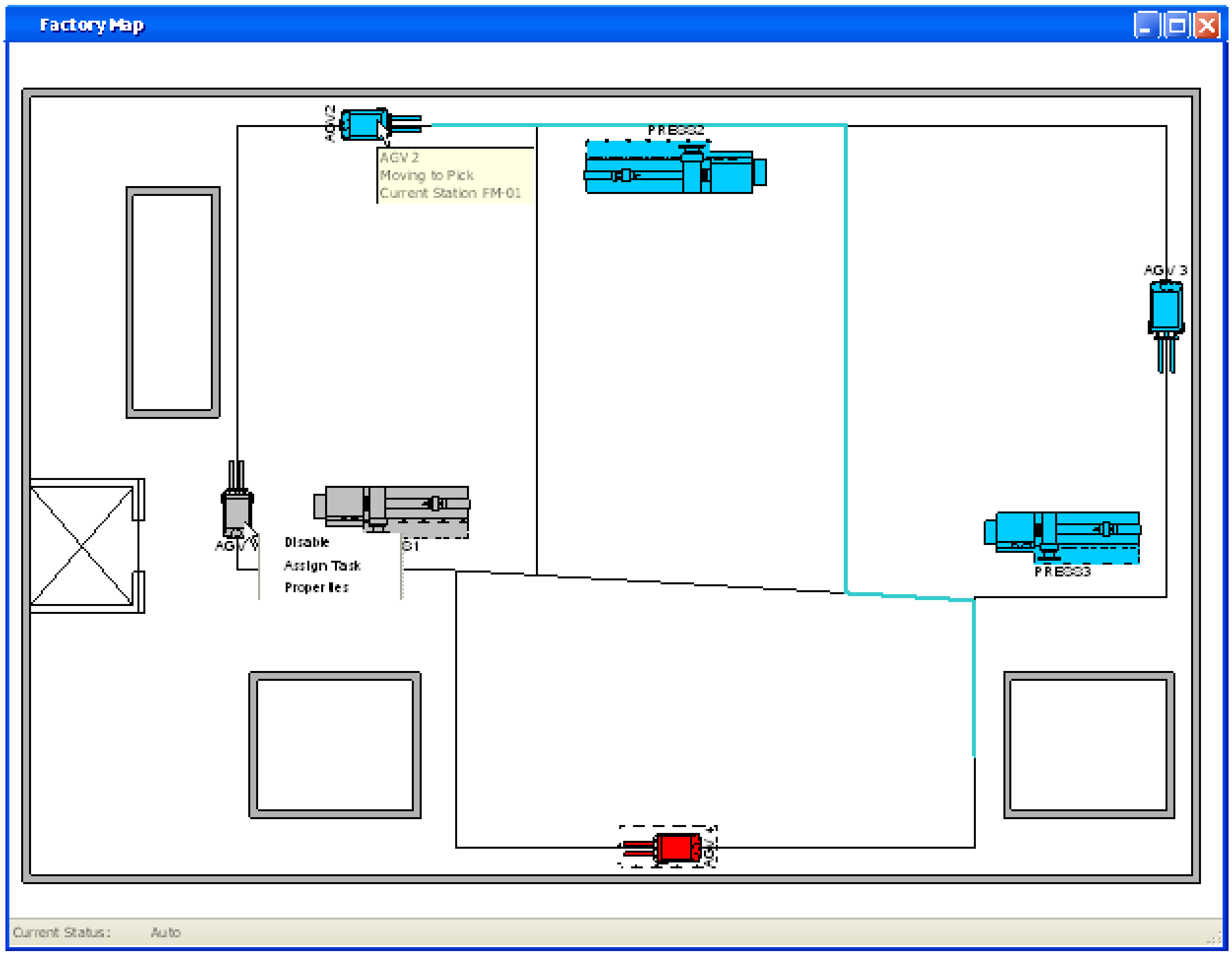
No	AGV	Pick-LVL	Received	Dispatch	Picked

Current Status: Auto

Map Form

This represents system status in map-style form. SYSTEM status is represented by color (gray – moving to load, blue – loaded, red - down). If SYSTEM is selected its route is highlighted on factory map. Among properties the form gives the ability to execute actions for each object via context menu. For example, right-clicking on SYSTEM we will get following list of possible actions:

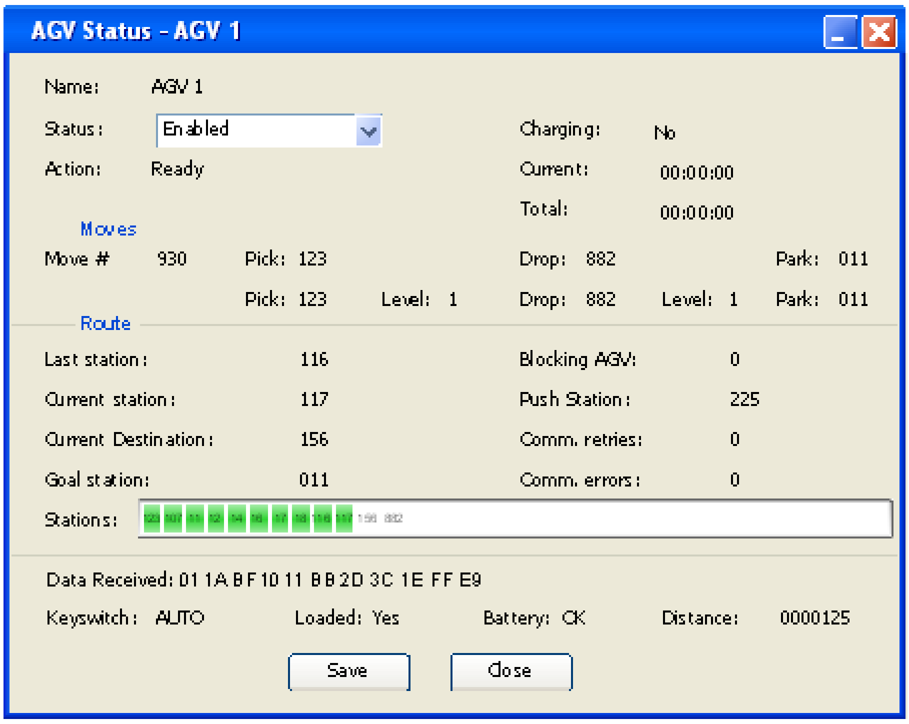
1. Enable
2. Disable
3. Remove from path
4. Go to charger
5. View status
6. View move orders



Current Status: Auto

SYSTEM Status Dialog

This dialog shows SYSTEM status and gives the ability to change the status for the privileged person.


AGV Status - AGV 1

Name: AGV 1

Status: Enabled

Action: Ready

Charging: No

Current: 00:00:00

Total: 00:00:00

Moves

Move # 930 Pick: 123 Drop: 882 Park: 011

Pick: 123 Level: 1 Drop: 882 Level: 1 Park: 011

Route

Last station: 116

Current station: 117

Current Destination: 156

Goal station: 011

Blocking AGV: 0

Push Station: 225

Comm. retries: 0

Comm. errors: 0

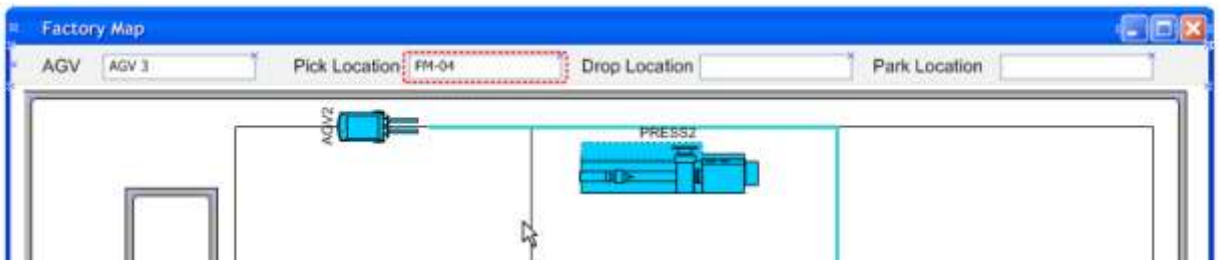
Stations: 123 107 111 112 114 116 117 118 116 117 156 882

Data Received: 01 1A B F 10 11 B B 2D 3C 1E FF E9

Keyswitch: AUTO Loaded: Yes Battery: OK Distance: 0000125

Setup SYSTEM   Path Toolbar

This purpose of this toolbar is to define SYSTEM's pick, drop and park location. The toolbar will be shown on the top of the map form and user will be able to select SYSTEM and stations.

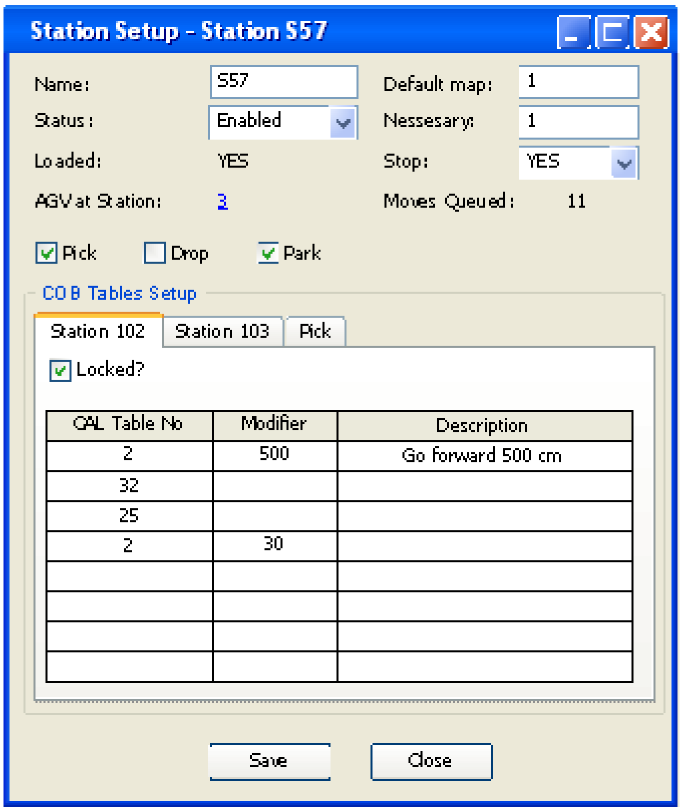


Factory Map

AGV: AGV 3 | Pick Location: FM-04 | Drop Location: | Park Location:

Station Setup Dialog

This dialog shows Station status and properties and gives the ability to change the status or some properties for the privileged person.



Station Setup - Station S57

Name:	S57	Default map:	1
Status:	Enabled	Nessesary:	1
Loaded:	YES	Stop:	YES
AGV at Station:	3	Moves Queued:	11

[✓] Pick [ ] Drop [✓] Park

COB Tables Setup

Station 102 | Station 103 | Pick

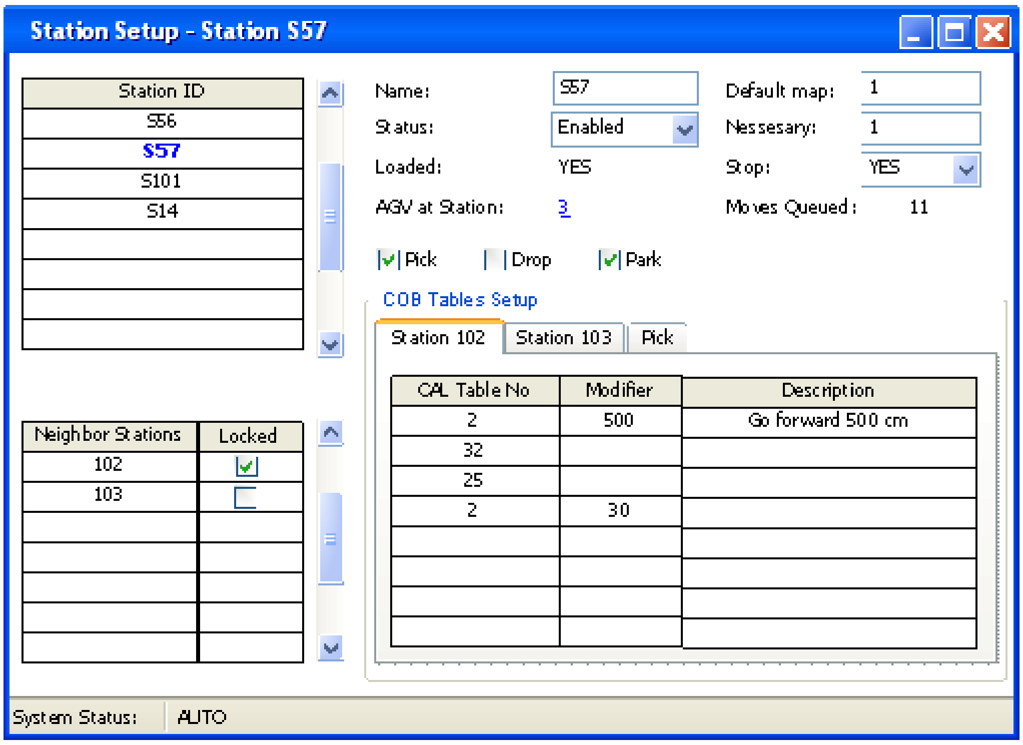
[✓] Locked?

CAL Table No	Modifier	Description
2	500	Go forward 500 cm
32		
25		
2	30	

Save Close

Station Setup Form

The Station Setup form is used during system maintains to change station properties.



Station Setup - Station S57

Station ID
S56
S57
S101
S14

Neighbor Stations	Locked
102	✓
103	

Name:	S57	Default map:	1
Status:	Enabled	Nessesary:	1
Loaded:	YES	Stop:	YES
AGV at Station:	3	Moves Queued:	11

[✓] Pick [ ] Drop [✓] Park

COB Tables Setup

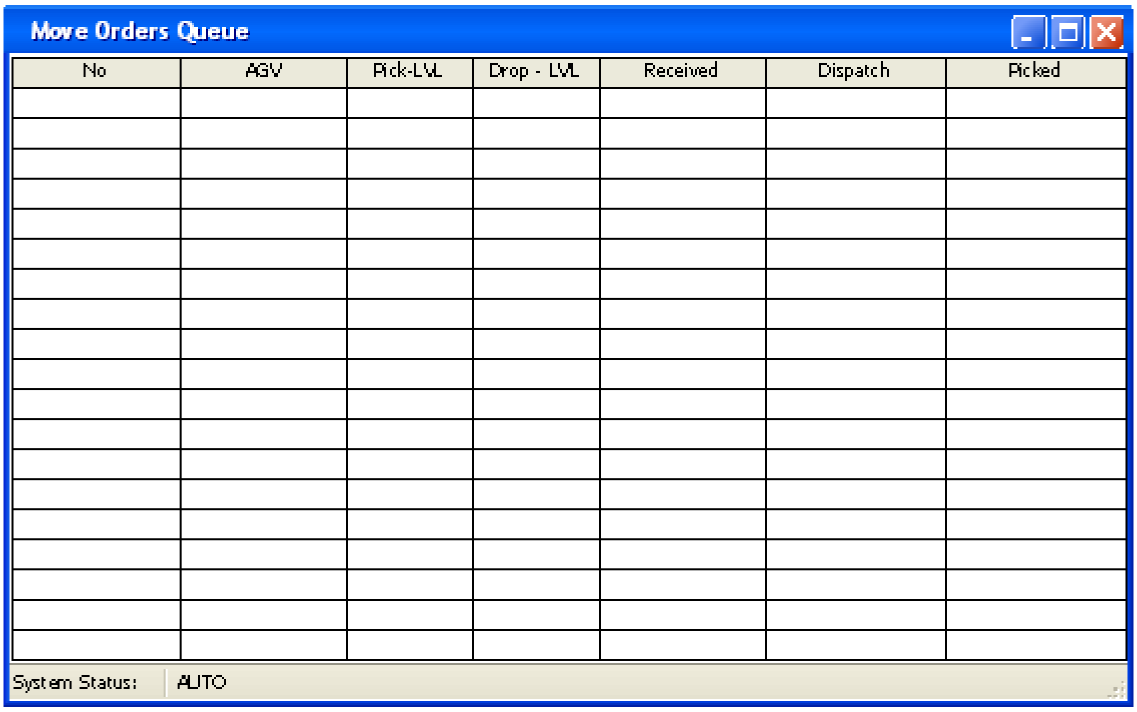
Station 102 | Station 103 | Pick

CAL Table No	Modifier	Description
2	500	Go forward 500 cm
32		
25		
2	30	

System Status: AUTO

Move Orders Queue Form

This form shows the move orders queue with possibility to find, filter, sort and delete move orders. All Windows UI features will be enabled: sorting by any column, moving columns using mouse and select columns to show.

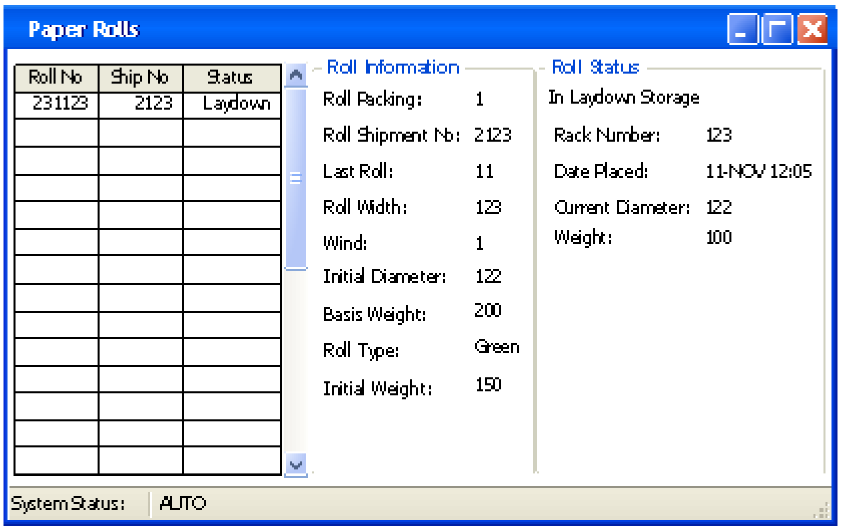


No	AGV	Pick-LVL	Drop - LVL	Received	Dispatch	Picked

System Status: AUTO

Paper Rolls Overview Form

This form displays information about all rolls used in system. This form gives the ability to find particular paper roll, to sort and filter rolls list and to waste particular roll. Roll Status can be color-coded. User can define set of columns in the left part of the form.


Paper Rolls

Roll No	Ship No	Status
231123	2123	Laydown

Roll Information

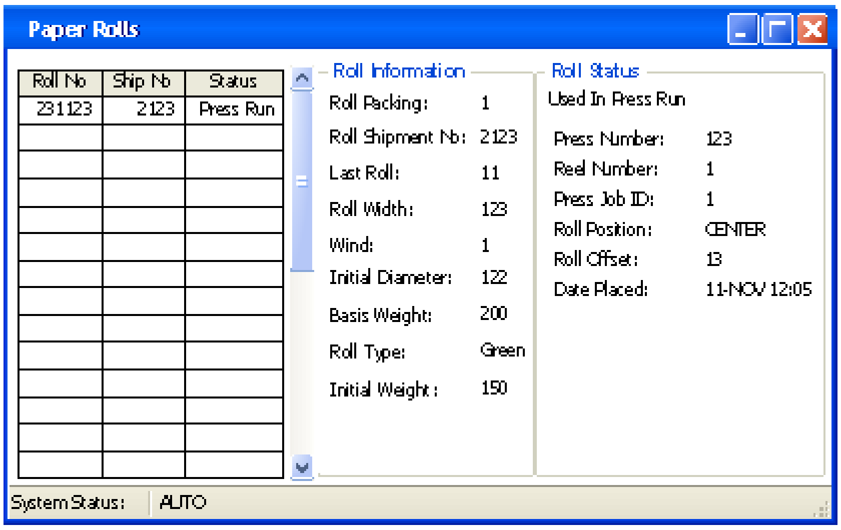
Roll Packing:	1
Roll Shipment No:	2123
Last Roll:	11
Roll Width:	123
Wind:	1
Initial Diameter:	122
Basis Weight:	200
Roll Type:	Green
Initial Weight:	150

Roll Status

In Laydown Storage

Rack Number:	123
Date Placed:	11-NOV 12:05
Current Diameter:	122
Weight:	100

System Status: AUTO


Paper Rolls

Roll No	Ship No	Status
231123	2123	Press Run

Roll Information

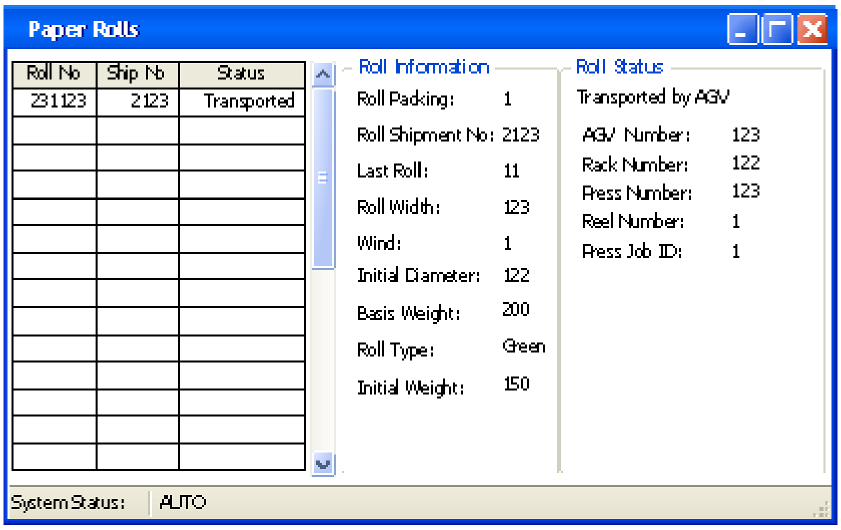
Roll Packing:	1
Roll Shipment No:	2123
Last Roll:	11
Roll Width:	123
Wind:	1
Initial Diameter:	122
Basis Weight:	200
Roll Type:	Green
Initial Weight :	150

Roll Status

Used In Press Run

Press Number:	123
Reel Number:	1
Press Job ID:	1
Roll Position:	CENTER
Roll Offset:	13
Date Placed:	11-NOV 12:05

System Status: AUTO



Paper Rolls

Roll No	Ship No	Status
231123	2123	Transported

Roll Information

Roll Packing:	1
Roll Shipment No:	2123
Last Roll:	11
Roll Width:	123
Wind:	1
Initial Diameter:	122
Basis Weight:	200
Roll Type:	Green
Initial Weight:	150

Roll Status

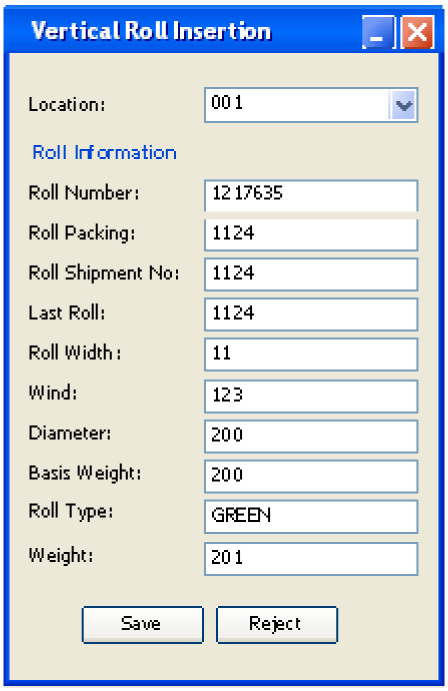
Transported by AGV

AGV Number:	123
Rack Number:	122
Press Number:	123
Reel Number:	1
Press Job ID:	1

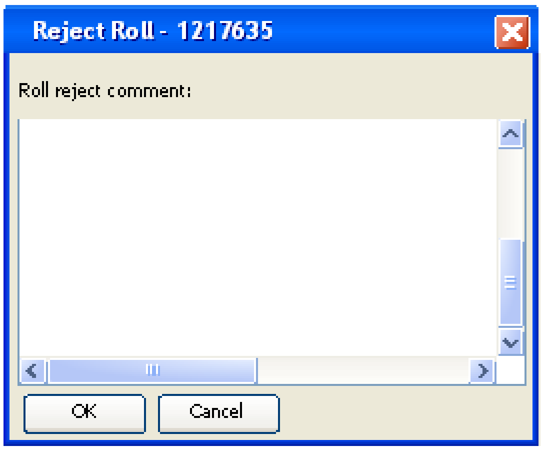
System Status: AUTO

Vertical Roll Insertion Dialog

This dialog is dedicated to person on Roll Preparation Station. When the roll is moved to the roll preparation station the operator will scan the bar code label to get the roll number. The new SYSTEM S will use this number to retrieve the roll information from the file containing the Arbitral inventory data manifest and fills in the remaining data fields. This form gives the ability to reject roll and write reject reason.


Vertical Roll Insertion

Location:	001
Roll Information	
Roll Number:	1217635
Roll Packing:	1124
Roll Shipment No:	1124
Last Roll:	1124
Roll Width:	11
Wind:	123
Diameter:	200
Basis Weight:	200
Roll Type:	GREEN
Weight:	201

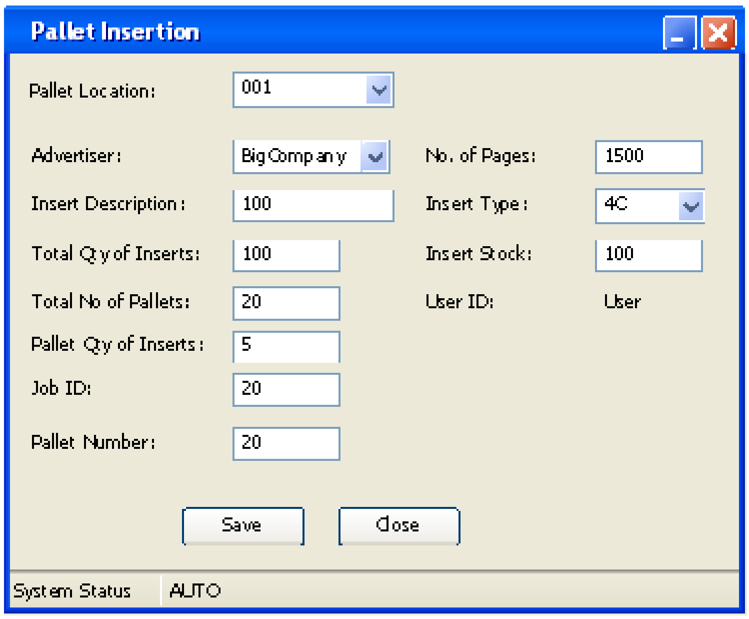

Reject Roll - 1217635

Roll reject comment:

Dividing form to two windows will prevent from accidental rejecting roll. Reject comment is a mandatory field.

Pallet Insertion Dialog

The operator uses this Dialog to enter the pallet information when it is first received and creates the initial entry into the Pallet Inventory Data Base for this specific pallet.
The new system will utilize features such as look ahead spelling, selection from pull down boxes and screen population from the data base when possible. It is decided to leave this form design as in the previous version of the system for easier of use.

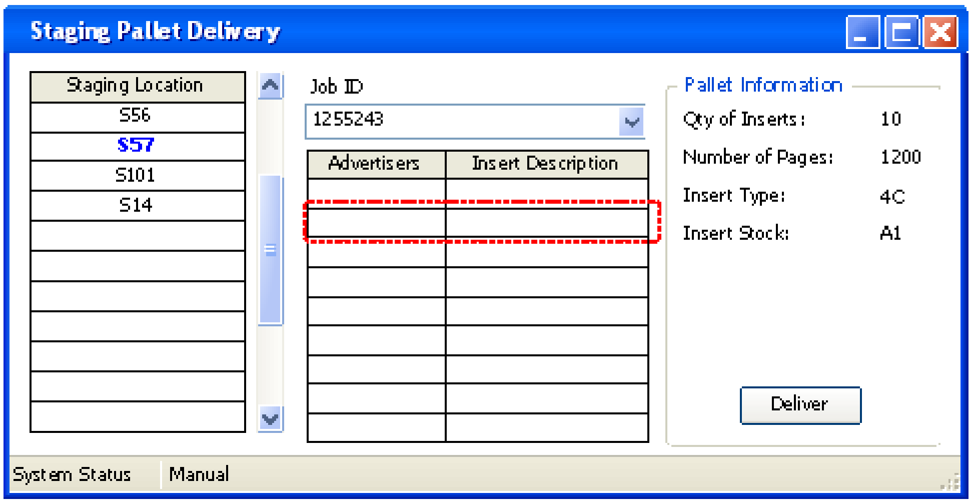


Pallet Location:	001		
Advertiser:	BigCompany	No. of Pages:	1500
Insert Description:	100	Insert Type:	4C
Total Qty of Inserts:	100	Insert Stock:	100
Total No of Pallets:	20	User ID:	User
Pallet Qty of Inserts:	5		
Job ID:	20		
Pallet Number:	20		

System Status AUTO

Staging Pallet Delivery Form

The form will allow the operator to select a staging location and enter (or select from drop-down list) the job number for each location.
The SYSTEM S will display all advertisers' records for each Job ID with the Insert description. The operator can select the desired advertisers and the system will display the number of inserts, insert type, and insert stock from the Data Base.
The operator can confirm the delivery request and the SYSTEM S will create a transport order for that pallet.


Staging Pallet Delivery

Staging Location
S56
S57
S101
S14

Job ID: 1255243

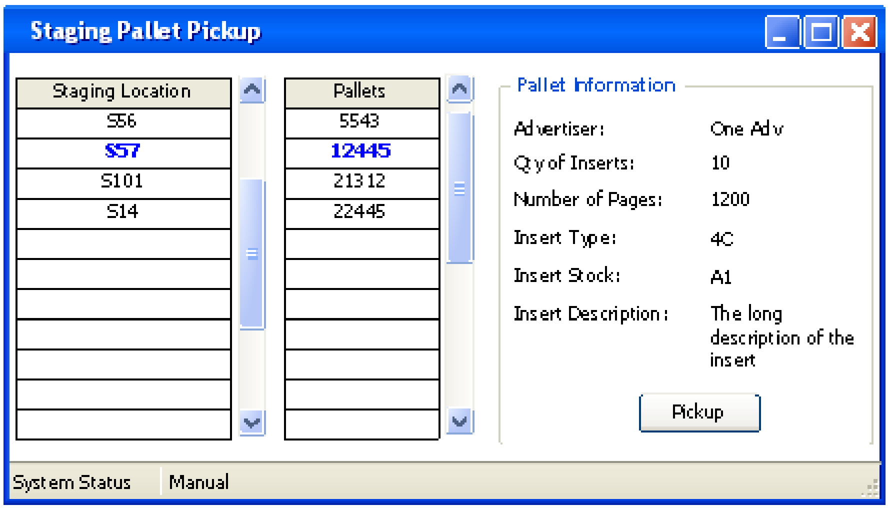
Advertisers	Insert Description

Pallet Information

Qty of Inserts:	10
Number of Pages:	1200
Insert Type:	4C
Insert Stock:	A1

Staging Pallet Pickup Form

If some pallets are unused the operator will use the staging pallet pickup form to notify the system to pick up the unused pallets. The system will display the staging location and the operator can select from pull down lists or use partial spelling to display the potential pallets to be retrieved (from the pallets delivered earlier that day) . Once the system has enough information it will populate the screen with the fields so the operator can confirm that this is the pallet for pickup. This confirmation initiates the transport order from the SYSTEM S control system.



Staging Location
S56
S57
S101
S14

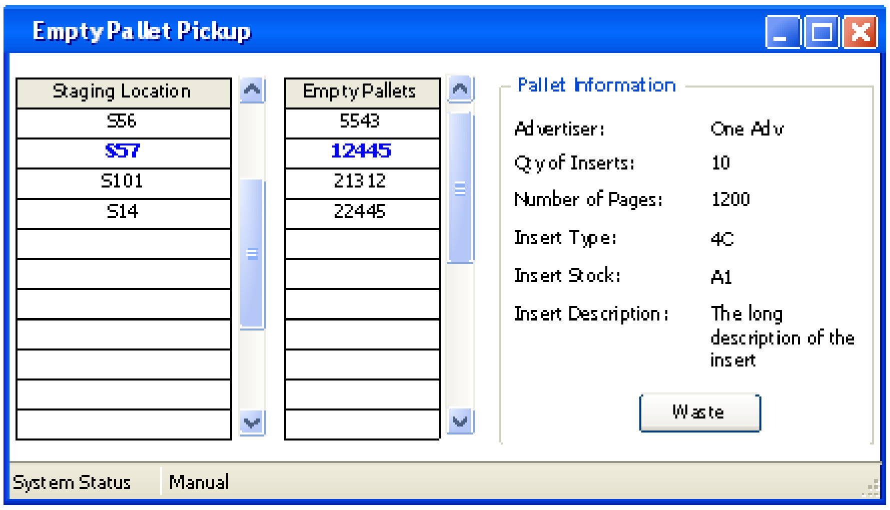
Pallets
5543
12445
21312
22445

Pallet Information

Advertiser:	One Adv
Qty of Inserts:	10
Number of Pages:	1200
Insert Type:	4C
Insert Stock:	A1
Insert Description:	The long description of the insert

Empty Pallet Pickup Form

This form provides a list of the Staging locations (PP01 through PP08); the operator will click on the locations that have empty pallets to be picked up. Operator can see the information about each pallet, if needed. This action will initiate a transport order by the SYSTEM S to pick up all empty pallets and transport to the Waste Area.



Staging Location	Empty Pallets
S56	5543
S57	**12445**
S101	21312
S14	22445

Pallet Information

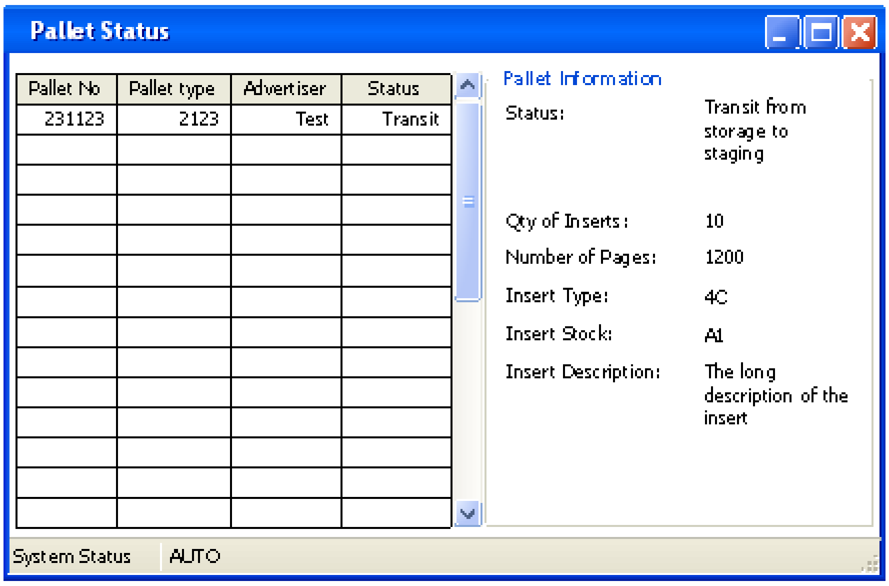
Advertiser:	One Adv
Qty of Inserts:	10
Number of Pages:	1200
Insert Type:	4C
Insert Stock:	A1
Insert Description:	The long description of the insert

System Status | Manual

Pallet Status Form

This form will allow the operator to select and display the status of the pallets by location or type of pallet. This screen will display all pallets requests made during a specified period (default last 24 hours). Latest Insertion request will be displayed first. Each pallet will be colored coded based on the status of the pallet. The most important 4 fields (to be defined) will be displayed in the list and the user can click on the Insertion Request for example to display all the information in the Pallet Data base associated with that record. The display types could be:

- Pallet in Receiving Rack location
- Pallet in transit from Receiving Rack to Pallet storage
- Pallet in Storage
- Pallet in transit from storage to staging location
- Pallets at staging



Pallet No	Pallet type	Advertiser	Status
231123	2123	Test	Transit

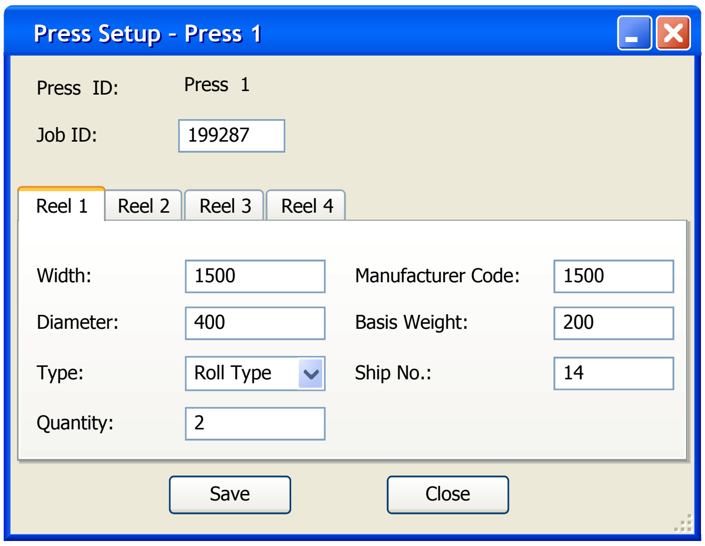
Pallet Information

Status:	Transit from storage to staging
Qty of Inserts:	10
Number of Pages:	1200
Insert Type:	4C
Insert Stock:	A1
Insert Description:	The long description of the insert

System Status AUTO

Press Setup Dialog

The Press Setup Dialog is used to specify the type of rolls to be delivered to a reel stand for a specified press run, or to modify an existing setup.

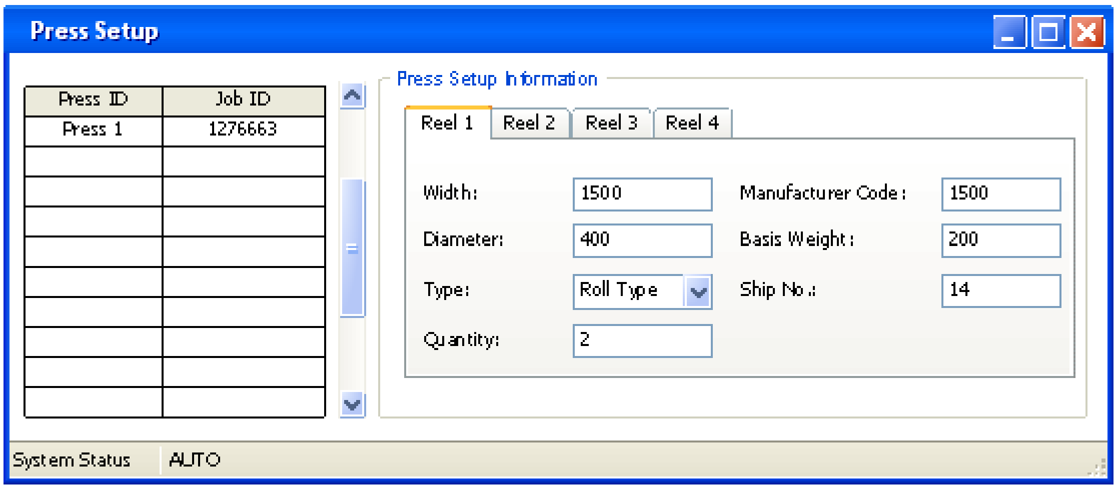


Press ID:	Press 1		
Job ID:	199287		
Width:	1500	Manufacturer Code:	1500
Diameter:	400	Basis Weight:	200
Type:	Roll Type	Ship No.:	14
Quantity:	2		

The form is planned for keyboard input. Navigation will look like this: Job ID – Reel 1 tab – Width – M. Code – Diameter – (so on from left to right up to down) – Reel 2 tab – (navigation like “Reel 1”).

Press Setup Form

The Press Setup Form is used to specify the type of rolls to be delivered to a reel stand for a specified press run, or to modify an existing setup. This form lets the user to select press from the list of all presses



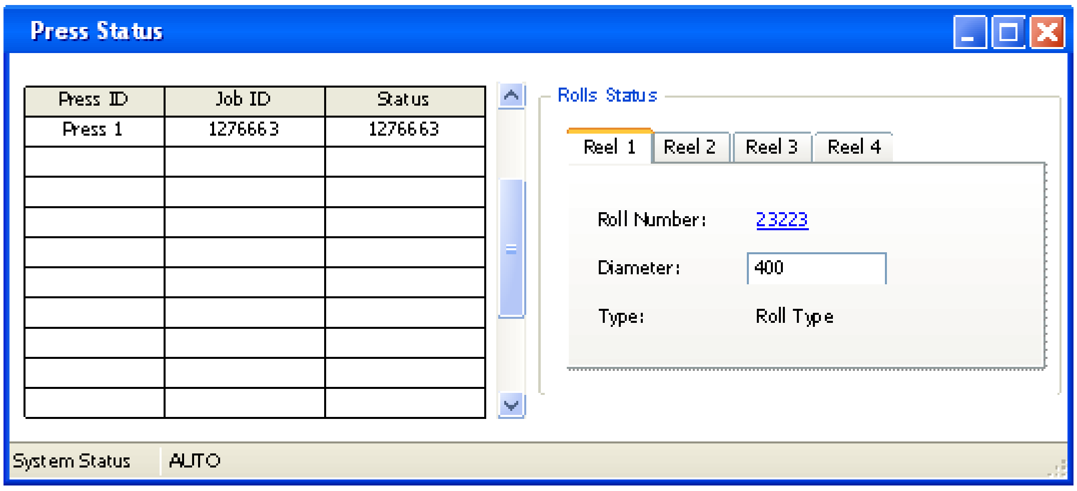
Press ID	Job ID
Press 1	1276663

Width:	1500	Manufacturer Code:	1500
Diameter:	400	Basis Weight:	200
Type:	Roll Type	Ship No.:	14
Quantity:	2		

System Status AUTO

Press Status Form

The Press Status form is used at the end of the press run to enter roll diameter for each roll on the reel. This form lets the user to select press from the list of all presses.



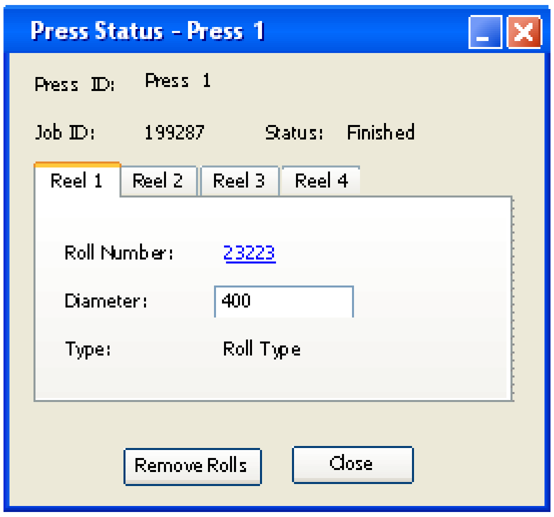
Press ID	Job ID	Status
Press 1	1276663	1276663

Roll Number:	23223
Diameter:	400
Type:	Roll Type

System Status AUTO

Press Status Dialog

The Press Status Dialog is used at the end of the press run to enter roll diameter for each roll on the reel.



Press ID: Press 1

Job ID: 199287 Status: Finished

Roll Number: 23223

Diameter: 400

Type: Roll Type

This form has only one field for enter data – “Diameter”. It’s a roll diameter after job finish. By clicking on “Roll Number” form with roll info will be shown.

Event Logging

For logging system standard Windows event logging system will be used. Event logging in Microsoft Windows provides a standard, centralized way for you to have your applications record important software and hardware events. When an error occurs, the system administrator or support technicians must determine what caused the error, attempt to recover any lost data, and prevent the error from recurring. It is helpful if applications, the operating system, and other system services record important events such as low-memory conditions or failed attempts to access a disk. The system administrator can use the event log to help determine what conditions caused the error and the context in which it occurred.

Windows supplies a standard user interface for viewing these event logs and a programming interface for examining log entries.

An *event,* as defined in Windows, is any significant occurrence — whether in the operating system or in an application — that requires users to be notified. Critical events are sent to the user in the form of an immediate message on the screen. Other event notifications are written to one of several *event logs* that record the information for future reference. Every event log entry is classified into one of the following categories: errors, warnings, information, success audits, or failure audits.

There are three event logs available by default:

- a System log, which tracks events that occur on system components (for example, a problem with a driver);
- a Security log, which tracks security changes and possible breaches;
- an Application log, which tracks events that occur in a registered application.

We can create your own custom logs using the language features in the **System.Diagnostics** namespace. And this log will be used for logging the SYSTEM.